Explanation in Information Systems: A Design Rationale Approach

Steven R. Haynes

Submitted for the Degree
of
Doctor of Philosophy

The London School of Economics and Political Science

Department of Information Systems
and
Department of Social Psychology

November 2001

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

This dissertation investigates the relationship between the information systems (IS) development context, and the context in which such systems are used. Misunderstandings and ambiguities emerge in the space between these contexts and often result in construction of systems that fail to meet the requirements and expectations of their intended users. This study explores this problem using an approach derived from three largely separate and distinct fields: explanation facilities in information systems, theories of explanation, and design rationale.

Explanation facilities are typically included in knowledge-based information systems, where their purpose is to provide system users with the underlying reasons for why the system reaches a particular conclusion or makes a particular recommendation. Prior research suggests that the presence of an explanation facility leads to increased acceptance of these conclusions and recommendations, therefore enhancing system usability. Theory of explanation is a field of study in which philosophers attempt to describe the unique nature of explanation and to identify criteria for explanation evaluation. Design rationale research is concerned with the capture, representation, and use of the deep domain and artefact knowledge that emerges from the design process. The design rationale approach goes beyond specification and suggests that to understand a system requires knowledge of the arguments that led to its realisation.

This study proposes a model of IS explanation structure and content derived from formal theories of explanation with a method for obtaining this content based on design rationale. The study has four goals: to derive a theory of explanation specific to the domain of information systems; to examine this definition empirically through a study involving IS development and management professionals; to investigate in a case study whether the information needed to populate the explanation model can be captured using design rationale techniques; and construction of prototype software to deliver explanations per the proposed framework.

Acknowledgements

I would first like to thank my thesis supervisors at the London School of Economics: Dr. Edgar Whitley from the Information Systems Department and Dr. Andy Wells from Social Psychology. Thank you for all your advice and support, and especially for your perseverance.

Dr. Chrisanthi Avgerou, Dr. Tony Cornford, and Dr. Jonathan Liebenau (Information Systems), and Dr. Bradley Franks and Dr. Patrick Humphreys (Social Psychology) all provided input and feedback at important points in the project. The assistance of many other members of the LSE Information Systems and Social Psychology faculties, as well that of fellow Ph.D. students in both departments was invaluable.

My new colleagues at the Pennsylvania State University School of Information Sciences and Technology have provided advice, support, and a lot of constructive prodding over the last year. I look forward to many future projects with them.

The confidence shown by my parents, my brothers and sisters, and by my extended family provided a special kind of motivation to get this work done, as always I am grateful to them.

Last and best thanks to Anne and Helena for their love and understanding.

Table of Contents

List of Figures

List of Tables

1 Introduction

This dissertation investigates the relationship between the information systems (IS) development context, and the context in which such systems are used. As systems become increasingly complex and pervasive, misunderstandings and ambiguities emerge in the space between these contexts and often result in the construction of systems that fail to meet the requirements and expectations of their intended users. One way commonly proposed to address this issue is to provide users with an integrated facility to *explain* the system and its operations. This study extends this idea using an approach derived from three largely separate and distinct fields: explanation facilities in information systems, theories of explanation, and design rationale.

Explanation facilities are typically included in knowledge-based information systems, where their purpose is to provide system users with the underlying reasons for why the system reaches a particular conclusion or makes a particular recommendation. Prior research suggests that the presence of an explanation facility leads to increased acceptance of these conclusions and recommendations, therefore enhancing system usability (Teach & Shortliffe, 1981; Hayes-Roth & Jacobstein, 1994; Dhaliwal & Benbasat, 1996). Theory of explanation is a field of study in which philosophers attempt to describe the unique nature of explanation and to identify criteria for explanation evaluation. Design rationale research is concerned with the capture, representation, and use of the deep domain and artefact knowledge that emerges from the design process. The design rationale approach goes beyond specification and suggests that to understand a system requires knowledge of the arguments that led to its realisation.

This study proposes a model of IS explanation structure and content derived from formal theories of explanation with a method for obtaining this content based on design rationale. The study has four goals: to derive a theory of explanation specific to the domain of information systems; to examine this definition empirically through a study involving IS development and management professionals; to investigate in a case study whether the information needed to populate the explanation model can be

captured using design rationale techniques; and the construction of prototype software to deliver explanations per the proposed framework.

The rest of this introductory chapter provides further background to the thesis topic, sets out the problem statement and research questions at its core, and gives an overview of the methodology that was used to achieve the research goals.

1.1 Background

The study of how people attempt to use and understand computers, the difficulties that they encounter when attempting to understand or use them, and the development of ways to improve their experience with them is important. Carroll (1987) points out two simple reasons for why the study of human-computer interaction (HCI) is significant: computers are pervasive, and people often find their use problematic. These are the basic problems with which this thesis is concerned and the motivation behind its development. Specifically, this thesis postulates that if a computer software system can provide explanations by reference to the rationale behind its design, the relationship between its structure and its purpose, how and why it performs certain tasks in certain ways, and where these tasks fit into the application domain, the system will be more effective in its use context. The problems that information system development organisations encounter when attempting to meet these challenges are also considered in light of the practical nature of the information systems development field. Acknowledging the cost-benefit equations that drive software development efforts and justifying the additional overhead of explanation facilities to software project decision-makers is essential to the acceptance of these ideas.

The architecture and content of a particular organisation's information system infrastructure is increasingly representative of both the organisation's intellectual capital, and its market capitalisation. Human actors in the social systems where they interact are said to "inscribe" much of their knowledge into the tools and systems that they construct and use, or configure and use (Latour & Woolgar, 1986; Bowker & Leigh Star, 1994; Hutchins, 1995). Of course, not all organisational knowledge is inscribed into artefacts. Significant knowledge is also possessed by those who build, maintain, and manage these systems, and is embedded in the work practices and social context where systems are used. Albert and Bradley (1997) suggest that there is

a significant trend in industry towards the dominance by expert knowledge workers of the critical information assets of their employing organisations. Research into the large IS development process suggests that on a given project only a small number of exceptional individuals truly understand the system architecture relative to the problem domain (Curtis, et al., 1988). These facts point to the growing importance of efforts to develop better models, methods, and organisational priorities centred on the explication, management, and reuse of information system-related intellectual capital.

In fields of design from VLSI (Very Large Scale Integration) to distributed enterprise resource planning software, it is acknowledged that almost all significant IS technologies have become too complex for any single designer to fully understand all facets of their operation (Stefik & Conway, 1982; Winograd, 1995; Kraut & Streeter, 1995). The Law of Requisite Variety, from the study of cybernetics, suggests that *any* successful system implementation will eventually outgrow the ability of its designers to understand it as its functionality is continually extended to meet the needs of the context in which it is used (Heylighen, 1992). On the user's side, the increasing complexity of software applications they use has led to corresponding growth in the costs associated with support and training (Heckerman, et al., 1995). To manage effectively our implementation and use of these technologies requires new approaches to how we come to understand them.

1.1.1 Theoretical Framework

The theoretical integration undertaken in this thesis involves incorporating research into explanation of information systems, theories of explanation from philosophy, and the design rationale approach to IS development, into a model that addresses some of the key problems with capturing and communicating system understanding.

Explanation in Systems

The MYCIN project at Stanford University in the 1970s and early 1980s was one of the first knowledge-based system development efforts to take as an explicit goal system-generated explanations of the application's behaviour (Buchanan & Shortliffe, 1984). MYCIN was an expert system designed to support physicians in the diagnosis and treatment of infectious disease. Part of the MYCIN project was an attempt by members of the research team to adapt the MYCIN programs and knowledge base for

use as a tutoring aid for medical students (Clancey, 1983). In attempting this adaptation, Clancey (1983) found that the relatively simplistic, rule-trace approach to explanation employed by MYCIN left implicit much of the knowledge used to develop the production rules used by the system. This implicit knowledge, including the rationale behind rule development and the structure of rule order and rule dependencies, represented the critical information needed in order for MYCIN to truly explain its behaviour. The framework constructed by Clancey to inform the development of an improved explanation facility for MYCIN consisted of three types of information, classified as strategy, structure, and support. Clancey's work represents one of the earliest examples of investigations into the role of design rationales in system explanation. This and subsequent related work will be explored in greater detail in the chapters that follow.

Theories of Explanation

Pitt (1988) argues that one of the primary reasons that philosophers are concerned with the concept of explanation is because:

> *"...explanations should tell us how things work and thus allow us to manipulate our environment."*

For many people, the information systems that they use represent a significant proportion of their work environment. Information systems developers should be concerned with providing explanation facilities for the reasons alluded to by Pitt. "Tell us how things work" and "allow us to manipulate our environment" are two goals worth pursuing for those who contribute to the task of helping people use information systems, whether they are developers of online help systems, system manuals and other documentation, or even integrated explanation facilities.

Design Rationale

Among the definitions of design rationale provided by Moran and Carroll (1996) is:

> *"An expression of the relationships between a designed artifact, its purpose, the designer's conceptualisation, and the contextual constraints on realizing the purpose." (Moran and Carroll, 1996, p. 8)*

Another that they provide is:

"An explanation of why a designed artifact (or some feature of an artifact) is the way it is." (Moran and Carroll, 1996, p. 8)

Research into design rationale techniques and tools considers many different aspects of design rationale capture, representation, and use. Lee (1997) classifies research into design rationale based on several general questions including: what services should design rationale provide, how should design rationales be captured and accessed, how much design rationale should be represented explicitly, and how can they be managed in a cost-effective manner? Though one aspect of design rationale use often considered in the literature is its role as a learning tool for design teams within information systems organisations, prior research has not been extended to explore how design rationale can support users in their attempts to learn and understand the systems that are then provided. This thesis attempts to help address this gap by employing an explanation framework to organise design rationale knowledge for these system users.

The basic arguments put forth in this thesis, and the problems they are intended to address, are supported in the literature of HCI and information systems development - though in most cases without direct reference to the applicability of explanation systems as contributing to a potential solution. For example, Rettig (1992) argues that an additional requirement should be added to every specification of requirements produced for a software information system. This additional requirement is simple – it is "to facilitate understanding". In this thesis it is argued that an integrated explanation facility can play a key role in the comprehensibility and therefore the usability of information systems. Since the quality of an information system is increasingly being measured by its usability as expressed by a propensity to use (Davis, 1989; Keil, et al., 1995), explanation facilities can play a key role in the overall quality of these systems.

Researchers in the field of expert systems explanation have recognised the need to provide rationales for their systems recommendations. In many cases users of their systems needed to know *why* a system made a particular recommendation (Swartout, et al., 1991). Though the notion of knowledge engineering to support system explanations is typically confined to the sub-field of artificial intelligence, some researchers have pointed out that all information systems "traffic" in some kind of knowledge and must engage in some level of knowledge management to be at all

useful (Chandrasekaran, et al., 1999). Data structures and definitions, database models, and program specifications all act to define the ontology of an information system. However, findings from foundational research on expert systems, knowledge acquisition and engineering, and other AI topics have been largely left out of the discussions in analogous fields that are currently in vogue, such as knowledge management and organisational memory (Gregor & Benbasat, 1999).

Modern, off-the-shelf, consumer software such as Microsoft Word represent essentially knowledge-based systems; in the case of Word, one that provides recommendations in the form of templates and customisable command sets in the domain of writing and document production. The products of mainstream, commercial software development are increasingly representative of a blurring of the line between traditional management information systems (MIS) and artificial intelligence (Hedberg, 1998). Many researchers argue that essential differences exist between more generic information systems and those commonly classified as expert systems (Yoon, et al., 1995). However, one of the most common criteria identified as designating this difference is the existence of an explanation facility (Lamberti & Wallace, 1990). The complexity of modern consumer software applications, it is argued in this thesis, generally equals or surpasses that of the knowledge-based systems of the 1970s and 1980s that motivated the development of integrated explanation facilities (IEFs) in software. Yet IEFs are still considered a system feature with a utility limited to the domain of knowledge-based systems, rather than to all information systems.

1.2 Problem Domain

The objective of this research is to develop a model of explanation specific to the domain of information systems, to identify the elemental components of information systems explanations, and to explore how the information needed to provide these explanations might be obtained. The role of explanation in system and device understanding is acknowledged in the information systems research communities, but has been limited to a subset of systems commonly identified as knowledge-based or expert systems. One of the central suppositions of this thesis is that the concept of explanation deserves a more central position in the study of how information systems are constructed and used. Target beneficiaries of these explanations include the

system development team, other system developers tasked with maintaining the system or with reusing system components, system users, and other stakeholders - for example, managers, marketers, and attorneys (see Pouloudi & Whitley, 1997).

Much of the research that has been done into the nature and role of explanation in the information systems context has relied on relatively narrow, *ad hoc*, or ambiguous definitions of the explanation concept. A common starting point for much of this work is a definition of explanation drawn from a dictionary of English (e.g., Gregor & Benbasat, 1999; Ye & Johnson, 1995; Eberhart, 1995). In this thesis it is proposed that by analysing the central theories of explanation developed in philosophy and relating the results of this analysis to empirical work on the factors promoting system understanding, an idealised framework for what it means to explain an information system may be developed. The purpose of developing this explanation framework is to provide a set of baseline guidelines to inform the construction of information structures (such as data and database models), methods and tools for capturing this information in the IS development context, and delivery mechanisms (integrated explanation facilities) for providing explanations from within IS software.

Information systems research into the nature of explanation content and how that content might be obtained is less well represented in the literature than that which focuses on how humans and computers interact in an explanatory dialogue. A first step into the study of explanation of information systems is the development of a theoretically and empirically grounded conception of what an explanation is in this domain. While it seems true that much of the power and fluidity of human-to-human explanation derives from our ability to construct, evolve, and repair explanations 'on the fly' based on sentence fragments, slight verbal cues, and nuances of gesture, it seems unfortunately also true that significant barriers exist to prevent the design of computers and software systems with similar aptitudes. Human-to-human modes of communication do not often make good models for human-computer interaction (Shneiderman, 2000). If this is the case, perhaps our efforts are best directed to exploiting the relative power that computerised systems do provide, such as consistent information storage and fast retrieval regardless of extraneous factors (U.S. Department of Defence, 1987), and by investing in ways to leverage this power, such as focusing on how approaches such as hypertext and data design can be extended to facilitate a human-*computer* explanatory dialogue.

According to what is arguably the most influential theory of explanation in the philosophy of science, explanations are a form of *argument* where the components of the argument act to convince the receiver that an entity exists or that an event occurred because of a particular set of *reasons* (Hempel & Oppenheim, 1948). A design rationale is "the argument behind the artefact" (MacLean, et al., 1989), a formal or semi-formal representation of the design questions, the alternatives generated to 'solve' each of the questions in a design space, and the design criteria applied in choosing between these alternatives. The idea that design rationales might contribute to the ability to explain a system or device has been suggested in the literature (Gruber, 1991; Gruber & Russell, 1996; Shipman & McCall, 1997). This suggestion forms the basis of the central questions addressed by this thesis.

Central to the problem of building software systems capable of providing explanations are the following issues:

> *How do we explicitly map the structure and content of an information system to its purpose and context in order to provide coherent, sufficient explanations?*

and

> *How do we capture and organise the information necessary to construct these explanations, integrate this information into the structure of the system itself, and communicate these explanations to the system users?*

and

> *How do we facilitate the integration of explanation capabilities into modern information systems development practice; and can the cost of this integration be justified?*

1.2.1 Research Questions

Explanation is a significant topic of information systems research in the main, in other words, beyond the domain of knowledge-based and expert systems to which it has been largely confined. The discussion so far leads to the central research questions of the thesis. The first of these is:

Q1. What can philosophical theories of explanation contribute to the development of a framework for integrated explanation facilities?

Though the philosophical literature includes theorising on the content, structure, and purpose of explanations over the last several millennia, at least since Plato (Ruben, 1990), very little of this work has been referenced in the research on IEFs. Much of the most recent IEF work has focused on the mechanics of explanatory dialogues (e.g., Moore, 1995; Cawsey, 1992), but this work generally fails to provide a model of the knowledge content of explanation, focusing instead on the enormous challenges entailed by attempting to engineer software with explanation provision aptitudes analogous to those possessed by humans. This thesis includes a survey of philosophical work on theoretical explanation, in Chapter 3, and attempts to derive from this work a more well-defined notion of what it means to explanation in IS.

The second research question is:

Q2. Can ideas from the fields of explanation systems and design rationale be integrated into the framework from Q1 to produce an implementable model for explanations of information systems?

This question addresses the central exploratory and theoretical components of the thesis. In attempting to answer this question, research from the distinct areas of explanation theory, IEFs in information systems, design rationale, and a number of associated fields are reviewed to lay a foundation for the approach. A conceptual framework is developed to help operationalise the explanatory elements, and a multi-part, sequential research methodology is employed to explore the applicability as well as the strengths and weaknesses of the framework.

The third and final research question is:

Q3. Is the model from Q2 operationally realistic, is it cost-effective, and can it be integrated into the IS development process?

This question addresses the practical component of the thesis by asking whether the model developed in response to Q1 and Q2 can be implemented given the constraints of the organisational context. The analysis and design phases of an information system development or configuration project typically represent at least half the cost

of the entire project (Elam, et al., 1991). Given that the process of capturing design rationale adds yet another cost element to these phases of an IS project, these costs need to be justified in order for the ideas presented here to be acceptable in practice.

In this thesis, I derive a theoretical framework from formal theories of explanation, developments in integrated explanation facilities, and the field of design rationale and apply it to the development of information systems with the goal being to build software systems whose structure, function, purpose, and operation is made more apparent to the users. The following diagram provides a high-level overview of the thesis context and focal points.

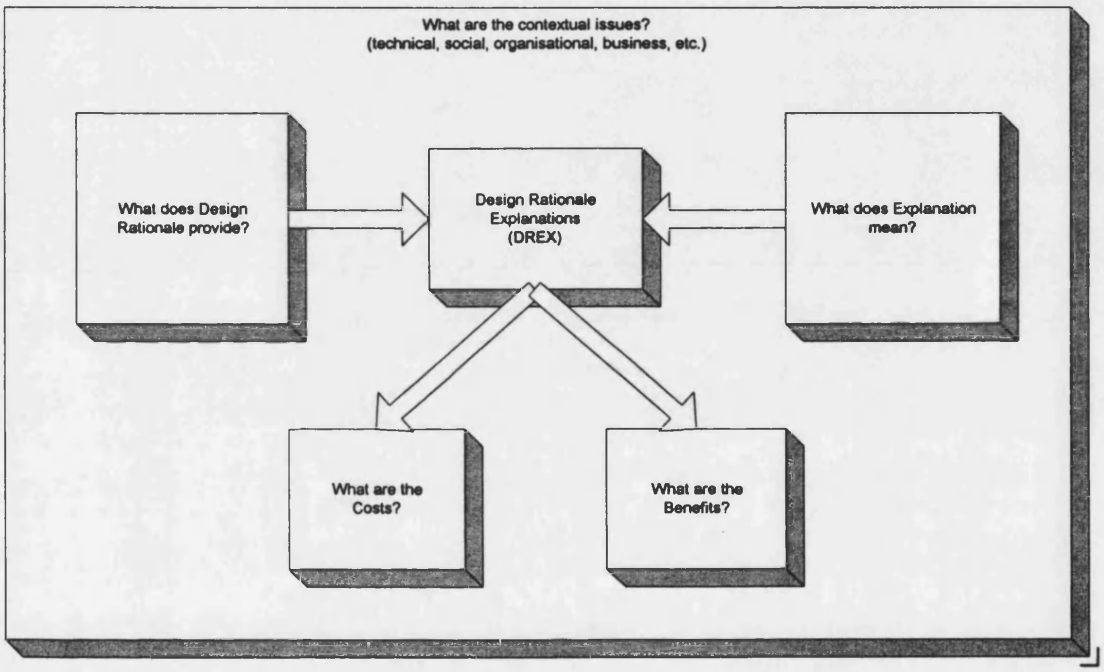


Figure 1 - Thesis Context and Focal Points

1.3 Significance of the Study

The ACM curriculum for the study of human-computer interaction identifies a number of key factors that will impact the future HCI design and engineering (ACM, 1992). First among these is that as IS becomes more pervasive and more complex, users will not have time learn these systems using traditional materials such as printed manuals. Their recommendations suggest that the importance of domain knowledge will increasingly outweigh that of programming knowledge as users seek a better fit between the systems that they use and the domain problems to which they are applied. As more and more tasks are computerized, organisations will seek to gain greater

efficiencies in the way in which workers are trained to use IS tools. Information systems that are able to explain their operation with reference to how their design maps to the application domain represent potentially fruitful means of addressing these issues.

Successful software product development depends on the quality of its human-interface design to bridge the gap between increasingly complex systems and their ever-widening target audience (Winograd, 1995; Dhaliwal & Benbasat, 1996). Norman (1986) describes the most significant problem of human-computer interaction and human-computer communication as the "Gulf of Understanding" that exists between the computer's understanding of the user, and the user's understanding of the computer. To address this problem, a number of questions must be answered including: what information can be used to fill this gulf, where does this information come from, how is it structured, how is it to be communicated, and what are the major challenges to the success of the resulting explanation product? Ultimately, attempts to include explanation facilities in information systems involve engineering human-computer interaction. An explanation-providing system embedded within an information system is expected to contribute to user-system fit and thus to facilitate users achieving their activity goals.

Johnson and Johnson (1993) describe current research into computer-generated explanations as characterised by several weaknesses. Among these is the lack of any unifying theory of explanation to act as a target normative model and as the basis for criteria to be applied in the evaluation of the explanation product. Though some recent work has made progress towards this goal (e.g., Gregor & Benbasat, 1999), gaps remain in the theoretical framework, especially in the area of explanation content. This thesis makes a contribution to a theory of explanation in the domain of information systems, focusing in particular on explanatory content, how this content may be sourced, and how explanations based on design rationale can be used to lessen the gap between the information systems development process and the information systems use context.

The study of human-computer interaction is a complex, interdisciplinary field that brings together computer and information scientists, psychologists, engineers, anthropologists, and researchers from other fields with the common goal of improving

people's experience with computers. In a series of studies designed to lay the foundation of usability studies at IBM, Carroll (1998) found that in their protocol analyses of users trying to learn computer systems with traditional documentation, they observed:

> *"...incredibly complex attributions, elaborately reasoned abductive inferences, and carefully performed, ritual behaviors."*

Though some software critics espouse a self-documenting approach to design (e.g., Rettig, 1991) and others argue for a minimalist approach to system documentation as an aid to exploratory learning (Carroll, 1990), it is unclear how either of these approaches scales up to larger, more complex and distributed software systems.

An integrated facility capable of providing system users with coherent, relevant explanations should be thought of as *necessary and integral* to an information system in any domain, rather than limited to artificial intelligence or computer-aided instruction applications. Explanations are *necessary and integral* to the users of the system because they support a fuller system understanding and therefore more effective use. Explanations are *necessary and integral* to the developers, owners, and other stakeholders of a software system because they support correct system design and maintainability, provide for the evolution of the system, and facilitate software component and design reuse.

1.4 Methodology

The motivations underpinning research projects vary widely and may include exploration, explanation, description, and prediction (Marshall and Rossman, 1995). The motivation behind the research reported in this dissertation is exploration, in particular, an exploration into the nature of IS explanations and the potential to source these explanations in the design rationale of the IS. Prior research into IEFs has largely ignored questions concerning the knowledge content of IS explanations, as opposed to their structure and delivery, and of how this explanation content might be obtained. Research into design rationale has not yet explicitly addressed the role of design rationale as a source of this explanatory content, and has not investigated the potential utility of design rationale information to end users of information systems. These are the target areas of the exploratory research reported here.

The field of information systems admits a broad range of research paradigms and research methods. This is perhaps a reflection of the wide variety of topics that fall under the field of IS: natural science, social science, design science, and philosophy are all practiced. This diversity is seen as both a strength and a weakness with regard to the continued viability of IS as an independent discipline (Robey, 1996). According to Robey, among the benefits derived from greater diversity of research methods and reference disciplines are: an expanded foundation for the knowledge claims made by IS researchers, the ability to attract quality researchers to the field through increased breadth of opportunity, heightened creativity, and advancing the cause of academic freedom.

Despite the benefits outlined above, Robey distinguishes between *disciplined* methodological pluralism and methodological anarchy and argues that researchers should choose methodological approaches not because they fit into a given paradigm or because of the researcher's belief in their intrinsic value, but because they are appropriate to the research task at hand. Others claim that by failing to present a cohesive, coherent 'story' to the wider academic community, information systems research risks being subsumed into one of the more well established disciplines - for example, computer science, management science, or organisational science - that is better able to articulate its objectives, approach, and methods (Benbasat &Weber, 1996). However, given the growth being experienced by the field of IS and the number of new departments, schools, and colleges being developed to meet the demand expressed both by industry and academia, IS as a discipline is likely to remain healthy and vibrant for the foreseeable future (Jarvenpaa, et al., 1991; Freeman, 2000).

1.4.1 Design Science

March and Smith (1995) classify research into information technology as one of two types: that with a natural science *intent* and that with a design science *intent.* They argue that research into information technologies with a natural science intent is *descriptive* and *knowledge-producing,* while research with a design science intent is *prescriptive* and *knowledge-using.* Simon (1996) describes the natural sciences as those concerned with how things are and how things work. Engineering sciences, on

the other hand, are concerned with artefacts, the properties of effective artefacts, and how to design them.

March and Smith argue that research with a natural science intent focuses on description. These descriptions form the basis of explanations which in turn are evaluated based on their ability to help formulate predictions (more on explanation/prediction symmetry in Chapter 3). The natural science intent is to obtain understanding, whereas the design science intent is concerned with achieving human goals by applying this understanding. An essential element of the design sciences is their emphasis on the normative and prescriptive, on improving artefacts through the development of better "should" and "ought" statements (Simon, 1996, p. 115).

According to Stefik and Conway (1982, p. 4), "knowledge is an artefact, worthy of design." By this statement they mean that information derived from the knowledge acquisition process may be structured in different ways to achieve different goals, in other words it can be engineered. Efforts to develop ontologies in the field of IS, for example, an ontology of IS explanation such as that attempted in this research, must continually focus on the utility of these efforts in practice (Chandrasekaran, et al., 1999). Design science involves balancing interests from a host of different perspectives including the social, psychological, and physical, in addition to technical factors, and acknowledging this balance is an essential aspect of the design process (Clancey, 1993b). The research reported in this dissertation has a design science intent, one that acknowledges information systems as a component of the socio-technical environment in which people work (Mumford, 1996; Hutchins, 1995).

1.4.2 Method Selection

The research approach assembled for this study is an example of "methodological pluralism" (Robey, 1996; Frechtling & Sharp, 1997; Miles & Huberman, 1994). Many researchers allow for such "methodological pluralism", whereas others claim that the tools selected by a researcher come burdened with a host of theory and value laden assumptions that make one set of tools incommensurate with the other (e.g., Strauss & Corbin, 1998; Frechtling & Sharp, 1997). Galliers and Land (1987) argue that the only significant measure of the value of information systems research is whether the work can be applied to improve practice. Another approach is that in assessing the validity and quality of a particular research work, focus should fall on

the details of how the data was collected, analysed, and presented rather than on the particular research ideology to which the research may be attributed (Glaser & Strauss, 1967; Strauss & Corbin, 1994).

Galliers and Land (1987) identified a set of methodological approaches suitable for research in information systems and organised these methods into a taxonomy that includes a mapping between the type of research being done, the object of study, and the method applied. A portion of this taxonomy, which includes the objects of study and research modes relevant to the methods represented in this dissertation, is shown below.

Table 1 - Information Systems Research Taxonomy

Research Mode	Object of Study		
	Organisation/ Group	Individual	Technology
Subjective/ Argument	Yes	Yes	Yes
Survey	Yes	Possibly	Possibly
Field Experiment	Yes	Yes	Yes
Laboratory Experiment	Possibly (small groups)	Yes	Yes

The objects of study analysed in this research include: groups engaged in information systems development (Organisation/Group); individuals interacting with technology in both development and user roles (Individual); and technology, specifically, technology to support the capture and management of design rationale and the use of the resulting knowledge base to construct explanations (Technology). The research modes represented in this dissertation include Subjective/Argumentative, in that a theoretical framework for the identification and evaluation of explanations in information systems is derived from the literature.

A form of *survey* is used in the first empirical part of the dissertation, where transcripts from interviews conducted with IS development professionals were collected and analysed to help ground the theory in practice, and both to focus and expand its conceptual basis. The *field experiment* component of the research strategy takes the form of a case study where an assembled team engaged in application

software design. Studies such as this one are typically described as taking place 'in the zoo', between laboratory and true field studies (MacLean, et al., 1996). The final empirical phase of the research, development of a software tool to construct and deliver design rationale explanations, fits their criteria for a laboratory experiment though it is more accurately classified as a proof-of-concept or build-and-evaluate prototype (March & Smith, 1995), in that its purpose is to show that such a facility may actually be implemented. Each of these methods is described in much greater detail in the chapters that follow.

Strauss and Corbin (1994) argue that while quantitative approaches are best suited to producing broad, generalisable results, they often suffer from a lack of depth in the results that they produce. Conversely, while qualitative approaches allow the researcher to explore a given topic at a deeper level, resource constraints usually prevent them from including a large number of study participants. Thus the generalisability of the results produced by qualitative methods is often called into question. The empirical components described in this thesis rely exclusively on qualitative methods. In the spirit of the design sciences, the use of qualitative methods represents a philosophical or epistemological commitment only to the extent that they were deemed useful and appropriate to the development of the thesis.

1.5 Thesis Structure

Following this introduction, Chapter 2 reviews research into IS explanations, including those found in expert systems, intelligent tutoring systems, application online help, and user manuals. This review presents the state-of-the-art in information system explanation. The objective of Chapter 2 is to derive a theoretical framework for explanations of information systems based on prior theoretical and empirical work in the field. This explanation framework and associated evaluation criteria will act as the baseline model of explanation as the thesis progresses.

Chapter 3 reviews the most influential theories from philosophical investigations into the nature of explanation. This review extends the framework developed in Chapter 2 and helps to establish a more theoretically grounded, baseline idea of what it means to explain an information system.

Chapter 4 sets out to describe the field of design rationale, its theoretical basis and history along with the techniques and tools that have been developed and applied in recent research. This chapter begins development of the argument for design rationale as an approach to capturing and representing IS explanation knowledge, one of the central arguments in the thesis.

Chapter 5 describes the first empirical component of the thesis. In this study, a series of semi-structured interviews was conducted with information systems professionals. These interviews were an attempt to further elaborate on the question of what constitutes an explanation in the information systems domain, how can this information be captured, and how can it be communicated to interested parties? The data collected in these interviews were analysed using qualitative techniques based on the grounded theory approach (Glaser & Strauss, 1967) and some of its more recent refinements (e.g., Miles & Huberman, 1994; Kelle, 1995). A discussion of the results of this analysis is included at the end of the chapter.

Chapter 6 reports on the second empirical component of the thesis, a case study of an information systems development project. The purpose of this study was to explore whether the information needed to populate the explanation model developed earlier in the thesis can be captured using techniques and tools of the design rationale approach. Tape recordings, design notes, meeting minutes, and drawings were used to construct a retrospective design rationale for the resulting information system. The content of this design rationale knowledge base is then analysed relative to the IS explanation framework developed earlier in the thesis. This chapter also describes an integrated explanation facility, a proof-of-concept software tool, which was constructed based on the principles developed throughout the thesis. This prototype was integrated into the information system designed and built as part of the case study described earlier in the chapter.

Chapter 7 is a general discussion of the results obtained from this multi-part study in light of the original research questions presented in Chapter 1. The extent to which the empirical data map to the theoretical integration described in chapters 2 through 5 is explored as are the various problems of data interpretation that were encountered in the analysis. The chapter concludes with a discussion of the opportunities for further research that emerged from this work.

2 Explaining Information Systems

This chapter begins an investigation into the research foundations upon which the thesis is built. The chapter is a review of work on integrated explanation facilities (IEF), focusing in particular on computer-generated explanation tools embedded in knowledge-based and expert systems, and to a lesser degree on potential contributions from the areas of intelligent tutoring systems (ITS) and information systems documentation including user manuals and online help systems. The objective of this chapter is to examine IEF research with a particular focus on the models of explanation employed by the most important research projects in the field. Chapters 3 and 4 examine theories of explanation and design rationale respectively, and theoretical integration of these topics is developed. The empirical studies described in chapters 5 and 6 then investigate this integration.

2.1 Explanation in Expert Systems

Expert systems are among the most mature and most commercially successful products of artificial intelligence research (Hayes-Roth & Jacobstein, 1994; Gill, 1995). Systems of this type are now used worldwide in a broad range of applications from blast furnace control in Japan to hurricane damage assessment in the Caribbean (e.g., Liebowitz, 1997; Wong & Monaco, 1995). Expert systems are knowledge-based applications that are designed to partially embody and augment human expertise within a well-bounded, protocol-oriented domain such as medicine or accountancy (Buchanan & Shortliffe, 1985; Edwards & Connell, 1989). Despite their success, it has been shown that significant problems arise when people seek to benefit from this computerised expertise in a manner consistent with their own abilities, experiences, and ways of working (Yoon, et al., 1995).

Computerised expertise is most often applied in two ways: to the solution of problems within a well-defined domain, and to imparting an understanding of a given field to those who do not possess this special knowledge. The latter process involves an expert providing an explanation of an entity or concept to another who seeks to *understand* what the expert tries to impart. Among the many descriptions of the role of an expert and of the nature of expertise is Winston's (1984):

"Human experts solve problems easily. They explain what they do. They judge the reliability of their own conclusions...".

This is a succinct account of what we expect from an expert, and a useful approximation of what we should expect from a computer program that is considered to embody expertise. What is less clear is the extent to which this definition and standard of expertise, especially as regards the *explanation* of problem solving processes and outcomes, is achieved by expert system developers in their work.

Virtually all descriptions of the nature and development of expert systems include an explanation facility among their core components. It is often argued that for a software system to be truly expert, it must have the ability to provide the user with an understanding of why and how it reaches a particular conclusion, therefore explanation facilities are considered integral to the functionality of an expert system (Davis, 1984; Sell, 1985, Swartout & Smoliar, 1987, Wick & Slagle, 1989). Most expert systems are designed to provide explanations for their reasoning in the form of elementary answers to "Why" and "How" questions posed by their users (Rich & Knight, 1991). Although the role of an expert system in instruction is considered secondary to its primary problem-solving duties, it has been argued that an important objective of these systems should be to impart knowledge to their users and therefore reduce redundant consultation sessions, especially in situations with a high cost, high risk or strict temporal element (Clancey, 1981; Berry & Broadbent, 1987). Inclusion of explanation facilities is one means by which an expert system can be made more effective at conveying the information required to achieve this goal and is a major factor in user acceptance of expert systems technologies (Hayes-Roth & Jacobstein, 1994; Whitley, 1990).

2.1.1 Why Provide Explanations?

Among the reasons that first gave rise to the inclusion of explanation facilities in expert systems were to help developers debug the system, to assure the knowledgeable user that a consultation session was on track, and to instruct the naive user in the problem domain (Berry and Broadbent, 1987; Hasling, et al., 1984; Lamberti & Wallace, 1990; Gill, 1995). One study concluded that not the expert advice given, but the ability to explain that advice, was the single most important factor in user acceptance of expert system applications (Teach & Shortliffe, 1981).

Explanations may also be useful during 'breakdowns' that occur in the user-system-task flow, such as when the software anomalies (bugs) occur or a user requires some specific information to complete a task (Gregor & Benbasat, 1999). Exclusive use of rule trace explanation facilities meant that first generation expert systems were able to provide only the first two of these functions. Rule trace explanations, while extremely helpful to the developer, often served to confuse more naive users when they were presented with an overload of detailed, arcane information in the language of the system development environment (Wick & Thompson, 1992; Berry & Broadbent, 1987). This information, if not harmful, does little more than describe the current state of the expert system rule base in terms that only its creators or an experienced programmer can comprehend, little justification or explanation of any use to the less knowledgeable user is provided.

Experts who do not possess the ability to draw on a base of increasingly detailed and experiential information in the course of providing an explanation are frequently the subject of credibility problems, especially in safety-critical domains or those that are considered to have a rich, intellectual content (Chandrasekaran & Mittal, 1983). It is often claimed that in order to successfully explain some concept at a level describable as expert, a clear model of the domain knowledge and a rigorous understanding of the problem solving process is obligatory (Hasling et al., 1984). However, the emerging field of situated cognition suggests that much of the expertise possessed by humans is embodied and contingent, therefore not necessarily readily available to explication (Schön, 1983; Suchman, 1987; Mirel, 1998).

Dhaliwal and Benbasat (1996) criticise the lack of a theoretical basis to support the proposition that knowledge-based systems should provide explanations and that users of these systems benefit from the use of these facilities. They argue that in addition to lacking an orienting theoretical framework, little empirical work has been done to examine the utility of IEFs in knowledge-based systems. They argue that the role of the expert system explanation facility is to clarify, teach, and convince. Users will not accept recommendations that emerge from reasoning that they do not understand. To be effective, an expert must provide both sound advice and the basis or rationale for that advice. They propose the following two questions as central to the development of a theoretical basis for explanation facilities:

Why should a knowledge-based system provide explanations?

What reasons do system users have for employing explanation facilities and what are the benefits that accrue to them when they do?

Increased attention to the explanation facility is a sign of a more user-centred approach to the development of expert systems. If expert systems are to become a well-accepted technology, more attention must be paid to the needs and wishes of the user community before, during, and after the expert system development lifecycle. In situations of user uncertainty or disagreement and in applications of computer aided instruction - particularly those of a safety-critical nature - the competency and coherence of the explanation facility that is provided with the expert system is one of its essential components, along with the reasoning engine and the knowledge-base.

Few empirical studies have been conducted to evaluate whether providing explanations improved the user's experience with a given system (Johnson & Johnson, 1993; Ye & Johnson 1995). However, one field survey of expert system development shell users reported that the existence of an explanation facility and the ability to customise explanations were among the most important evaluation criteria for these tools (Stylianou, et al., 1992). These features were ranked fourth out of 90 criteria and included the importance of explanation both to the developer of an expert system and to its end users. The authors of this report conclude that explanation facilities are crucial to confidence building among the end-user community as well as to the perception of control or empowerment that users of these systems obtain. Although some argue against providing such control to naive users or in safety critical systems due to the potential implications of misuse, this lack of control runs counter to widely accepted user-system interface development guide-lines (Berry & Broadbent, 1987, p. 18). The addition to modern expert systems of mixed-initiative type interfaces that give the user the option of either taking control of the session or being led by the system is one of the improvements that have helped to mitigate these problems (Cleal & Heaton, 1988, p. 21).

In another study concerned with the role of trust in expert systems' advice, researchers found that though explanations had a positive effect on user *agreement* with the advice given by these systems, explanations did not improve user confidence in the advice provided (Lerch, et al., 1997). The authors of this study caution however

that the system and explanations provided in this laboratory experiment were simplistic, and that a more significant effect might emerge with more complex systems, explanations and/or domains.

2.1.2 Foundations of Expert System Explanation

The following sections provide an overview of the most important research programmes related to explanations in expert systems, sometimes referred to more generically as knowledge-based systems (KBS). The focus of these sections is on their potential contributions to a more general theory of explanation for IS. Since expert systems can rely on the use of a more or less homogeneous data structure (the production rule) and homogeneous processing approach, the inference engine, when considering how explanations might be obtained, these projects provide only some of the pieces of the IS explanation puzzle. Still, the contributions they have provided to the study of computer-generated explanations remain the seminal standard for work such as that described here.

MYCIN

Integrated explanation facilities (IEFs) first emerged as a significant and independent topic of study from the MYCIN experiments at Stanford University in the 1970s and early 1980s (Buchanan & Shortliffe, 1984). The goal of the MYCIN project was to develop an expert system to support physicians engaged in diagnoses of bacterial infections. From the beginning of the project, the MYCIN team identified the ability to explain its problem-solving logic as a fundamental requirement of the system. In fact, explanation, along with knowledge acquisition, were identified as the two key problem areas in the development of the system (Buchanan & Shortliffe, 1984).

In the early MYCIN days of expert systems research, explanations of a system's reasoning were considered primarily as a resource to support system developers in debugging the reasoning process used by the system. The MYCIN team's original conception of explanation was to extend this notion so that users were provided with the reasoning steps, the relationship between a request for information and a system process, the relationship between system goals, and how particular system goals or outcomes are achieved (Clancey, 1983). However, the MYCIN development team realised early in the project that providing explanations contributed to the overall

acceptance of the system and facilitated deeper learning of the system domain by its users (Dhaliwal and Benbasat, 1996).

MYCIN is a backward chaining, goal-oriented expert system. This means that in the process of providing a diagnosis the user (or the system) first selects a hypothesis, or goal, and then attempts to confirm it by working backwards through the rules that lead to the conclusion suggested by the hypothesis. Conversely, an expert system employing a forward-chaining inference strategy begins with *a priori* facts, collects additional facts from user input, and infers new facts from these using the system's production rules. The forward-chaining process completes when one of the facts derived evaluates to some predefined goal state.

In MYCIN, the user can repeatedly query the systems conclusions by asking "Why?". A "Why?" question from the user causes MYCIN to ascend the goal tree and display each rule that was fired until the top-level goal is reached (recall that MYCIN uses a backward-chaining inference mechanism). In MYCIN "Why" is translated by the system as meaning 'How is this information useful to me?' The user may also ask "How", descending the goal tree from the conclusion through the various rules fired during the session (Bainbridge, 1988, p. 176). MYCIN's "How" corresponds to 'How did you arrive at this conclusion or rule?' Figure 2 below shows a series of MYCIN post-consultation explanations (from Clancey, 1983).

```
**DID YOU CONSIDER WHETHER THE PATIENT WAS RECEIVING STEROIDS?

Rule 543 COULD HAVE BEEN USED:

   whether J. Smith has received corticosteroids.

Clause 4 of RULE543 {the patient has received corticosteroids}
was already known to be false.

**HOW DID YOU KNOW THAT THE PATIENT HAD NOT RECEIVED STEROIDS?

RULE395 was used to conclude that J. Smith has not received
corticosteroids. The last question asked before the conclusion
was made was 23.

**WHAT WAS QUESTION 23?

Question 23 was asked in order to find out whether J. Smith is
a compromised host in an effort to execute RULE343.
```

Figure 2 - MYCIN Post-consultation Explanations

GUIDON

The GUIDON program was an attempt to reuse the inference procedures and rule base developed for MYCIN as the basis for a tutoring program in the program's domain of infectious disease. Since MYCIN's diagnostic performance was considered comparable to that of the faculty members at Stanford's medical school, it was hoped that the knowledge base and inference procedures used by the program could form the basis of tutoring system that could be made widely available to medical students at the school (Wenger, 1987). The insights that were gained through the development of GUIDON, NEOMYCIN (discussed below), and other MYCIN derivative projects represent a seminal contribution to the study of IEFs. Of particular importance is William Clancey's influential paper, *The Epistemology of a Rule-Based Expert System* (Clancey, 1983). The GUIDON program was William Clancey's thesis project at Stanford University (Clancey, 1986). The project involved analysing MYCIN's utility as a knowledge base for training applications as well as investigating the additional, generic functionality that would be needed to convert expert systems to educational tools. The strategy central to the effort was that inference procedures should be separated from the underlying knowledge base and that the knowledge base itself should be separated into two parts: domain specific (in this case, medical) knowledge and teaching, or pedagogical knowledge (Clancey, 1986). The hope central to this effort was that by separating teaching knowledge from domain specific knowledge, the program could be used as a tutoring device with any number of different, domain-specific knowledge bases.

In attempting to separate instructional knowledge from domain knowledge, and in analysing at a deep level what MYCIN was doing so that GUIDON could teach this material, Clancey discovered that much of the knowledge needed to explain how the system behaved was implicit in MYCIN's design, not explicitly represented in the application's knowledge base as was originally hoped. This insight led Clancey to develop a conceptual framework to help understand the nature of this implicit knowledge. The framework consisted of three categories of system knowledge described as: *structural*, *strategic*, and *support*.

Structural knowledge refers to information that was embedded in the ordering of MYCIN's production rule base, how the rules were indexed and categorised, and the relations between rules. GUIDON was unable to express much of the problem

structuring and problem solving approach employed by MYCIN since information was represented only as the order in which the rule clauses were designed to fire. In addition, many rules were found to exist whose purpose was simply to control the firing of other rules. The existence of these meta-rules emerged as a key component of the reasoning strategy that needed to be explained for the tutorial to be effective. Since MYCIN's knowledge engineers did not document the rationale behind rule clause ordering and this rule chaining structure, this information was not available to the GUIDON tutorial. This implicit knowledge of how different rules were combined and how the evaluation and firing of one set of rules led to the evaluation of others emerged as a crucial missing link when attempting to articulate the knowledge possessed in MYCIN (Wenger, 1987).

Clancey described the role of *structural* knowledge in an explanation framework as the bridge between the generic problem solving and knowledge representation strategies at the *strategic* level, and the domain specific hypotheses, goals, and rules of a particular knowledge-base. Structural knowledge is the way in which rules are classified and the methods that define how rules can be combined, or chained into a hierarchy or goal-directed order in the knowledge base. He felt that this structural knowledge acted as an index into the knowledge base and that it was used as a mechanism to populate the knowledge structure placeholders at the strategic level when they were operating on a particular problem. In another study, Lamberti and Wallace (1990) examined the relationship between the level of task uncertainty, the proficiency of the user (expert or novice), and the nature of the explanation content provided relative to experimental performance measures. They found that procedural knowledge was generally more useful because it supported both experts and novices as they attempted to form linkages or relationships between the elements of a given problem domain.

Clancey (1986) identifies the *strategic* knowledge required for system explanation as that which describes the methods by which certain rules are applied, the high-level problem solving approaches behind the inferences rules in the system. Production rules classifiable as strategic would typically be those that embodied the *heuristic* knowledge for which expert systems are most well known. This heuristic knowledge might include the higher-order goals of the system or a set of production rules, or the sequence in which a given set of rules was designed to fire (Ye & Johnson, 1995).

By *support* knowledge, Clancey was referring to the low-level, detailed information that was used to relate a rule to the underlying *causal process* in the world, the facts that justify the existence of a given rule. In the explanation framework developed by Clancey, support knowledge was used to *justify* a rule, to connect it to observed phenomena and empirically supported generalisations in the problem domain. The class of support knowledge was further sub-divided into four types of justifications: identification, causal, world fact, and domain fact. Identification knowledge is used to classify an entity or event, to show that an entity or event is an instance of a given concept or has properties that relate it to a concept. Causal knowledge refers to facts and arguments that may be linked together to demonstrate the mechanistic structure of a domain or sub-domain. These causal links may be well understood or may be only suggested by empirical evidence in the domain. World fact knowledge is characterised by Clancey as "common sense" knowledge about the world. He includes in this category what he calls *social patterns of behaviour*. His example describes an army recruit, whose living conditions in close proximity to others puts him at increased risk of a given disease (recall that MYCIN's domain is the diagnosis of infectious disease). The final category of knowledge used in the justification of a rule is domain facts, which are well-established heuristics that help with problem solving in a particular domain (e.g., to have been administered adequately, a drug must be detectable in the body at a certain concentration).

Clancey described the missing link between the high-level strategies and the domain-specific facts as *focusing principles* (Clancey, 1986). These focusing principles, which he felt could take the form of an argument or a justification, were thought to be crucial in attempting to teach novices in a domain since they connected ambiguous, high-level, problem-solving strategies with the facts of a single diagnostic scenario. He argues that an important aid to making these links was the causal chain behind the phenomenon to be explained. It describes the *knowledge roles* that particular pieces of information play in the explanation equation and how certain these pieces of information are combined to help make sense of an entity or an event. Clancey in many cases relies on the existence of underlying causal forces, and our ability to connect explanatory facts to them, to assist in the process of explanation.

NEOMYCIN

NEOMYCIN was an attempt to replace the domain-specific rule base used in MYCIN and in GUIDON with a generalised base of meta-rules seemingly applicable in a host of different domains (Clancey, 1983). The goal of the NEOMYCIN tutor was to extract a more generic tutor from the domain specific functionality of GUIDON, one that could teach more fundamental principles of medical problem solving and diagnosis (Clancey, 1986). In particular, NEOMYCIN was an attempt to focus on and exploit the structure and strategy components of the MYCIN and GUIDON knowledge bases, since, it was argued, these were seen to be higher order species of explanatory knowledge and ones that could be explored 'exhaustively' in contrast to what Clancey called 'the bottomless pit of support explanations' (Clancey, 1986, p. 46).

The NEOMYCIN team generated a set of guidelines regarding the development of an adequate expert system explanation facility that reflects their focus (from Hasling, 1984).

1. Explanations should not presuppose a particular user population.
2. Explanations should be informative; showing the rule numbers that fired is not enough.
3. Explanations should be concrete or abstract to suit the situation. The system should be able to explain a strategy and how it is applied.
4. Explanations should be useful for the designer as well as the user of the system.
5. Explanations should be possible at the lowest level of interest. This means that the "grain level" of the explanation knowledge should be fine enough to allow a good level of detail.

Clancey characterises the search for meta-rules among the rules of MYCIN as a search for rule *ordering patterns* (Clancey, 1993). He notes, however, that different applications might require different perspectives on the same rule base, so efforts at identification of appropriate ordering strategies need to be mindful of both existing

and future users and uses of the application. One of the concepts explicated in the NEOMYCIN effort was that strategies such as 'ask general questions first' and 'don't commit too early to a diagnostic path' were the kind of abstract approaches that could be generalised beyond the medical domain and were thus potentially useful when considering how the program could be used as a generic tutor (Clancey, 1986).

The results of the NEOMYCIN project helped to underscore some of the benefits that could be accrued through a more thoughtful and directed approach to the development of an explanation facility. In particular, two elements of the NEOMYCIN project, explicit representation of the problem solving strategy and the use of meta-rules in explanation planning, are discussed in detail later in the thesis.

The Explainable Expert System

The MYCIN project and its derivatives, as well as other expert systems development projects of the 1970s and early 1980s, can be viewed as the first generation of expert systems with IEFs. The XPLAIN system (Swartout, 1983), and the Explainable Expert System (EES) project at the University of Southern California are representative of the second generation of these types of systems (Swartout and Smoliar, 1987; Swartout, et al., 1991; Moore, 1995). The EES project was an attempt to construct IEFs with two explicit goals. The first of these was to recognise the interactive nature of human-to-human explanatory dialogue and to replicate this dialogue using explicit representations, called explanation plans, of potential machine-to-human explanations. This goal resulted in the development of an entirely separate stream of IEF research related to explanatory discourse management and is reviewed in several monographs including Moore (1995) and Cawsey (1992). Achieving natural language capabilities between humans and machines is a fascinating topic with a host of technical, psychological, and philosophical issues that must be addressed to achieve success. The scope of these issues precludes including them here, and this dissertation will not discuss the explanatory discourse management aspects of EES or of other IEF projects beyond a short review in section 2.3. Later chapters will however include a proposed method of delivering IEF explanations based on hypertext.

The second goal of the EES project, one central to the ideas developed in this thesis, was to provide explanations of an expert system by reference to the design rationales

which underlie the architecture of a system's operations and knowledge-base (Swartout, et al., 1991). While rule-trace explanations such as those provided by MYCIN exposed the behaviour of an expert system, they do not provide justifications for why a particular program behaviour is the correct approach in a particular use context (Swartout, 1983). Among the factors claimed by the EES team as contributing to 'good' explanations are these justifications for system behaviours plus descriptions of general problem solving and design strategies along with definitions of the terms used in the system domain. As on the MYCIN project, one of the challenges faced by the EES developers was to elucidate the link between domain-independent, strategic concepts and the domain-specific or instance specific information that is needed to apply a strategic concept in a particular scenario. For example, a design principle such as 'simplify wherever possible' might be instantiated in a software design as 'we can combine these two modules into one with no loss of cohesion'.

The EES attempts to solve this linkage problem through the concept of *capability descriptions*, which relate system goals to operationalised plans to achieve those goals. General problem-solving principles are represented by plans in the EES framework. Capability descriptions are used to define what the plan does, its competencies; they act as a sort of resume for their associated plans. System goals are mapped to plans and associated methods used to achieve those goals through these capability descriptions. Consider an EES-based expert system to support IS design. The knowledge-base goal 'simplify design' would be used to find a plan with a capability description of 'simplify design'. This plan would then have associated methods, such as analyse modular cohesion, for determining areas in which a design might be simplified. This mapping is used in reverse to provide explanations for EES behaviour.

Two techniques are described for matching specific goals to general system plans, specialisation and reformulation. Specialisation involves using variables in place of specific terms in a plan's capability description. For example, a capability description 'simplify X' might map to several potential goal values such as simplify module structure, simplify inheritance hierarchy, etc. This mapping is used when the system attempts to explain why a general design principle was employed in a particular instance. The reformulation principle is somewhat more difficult to picture in operational system terms. EES was designed to 'understand' the goals that it might be

called upon to explain. By understanding, the EES team apparently meant the ability to relate concepts that appear in a goal but not in a capability description to analogous concepts that do appear in some capability descriptions. This ability to construct a set of numerous explanations or hypotheses regarding a particular phenomenon from a fixed set - from the point of view of exposure to additional information - of facts and 'rules' is based on the creative mechanisms in the explanation process (Shank, 1986). Shank's view is that humans use "indices" into the mind to relate current episodes to past experiences, even when these past experiences are only tenuously related, and this ability is key to explaining new concepts. While the creative use of analogy would indeed help to solve many of the problems inherent in IEFs, it is difficult to imagine how the underlying knowledge base needed to support this system behaviour would be structured, and how much it would cost to be acquired and maintained. These problems are not addressed in the published EES results.

The EES is interesting because it includes a program writing application for automatic generation of expert systems for a given a target domain (Swartout and Smoliar, 1987). This application records the design 'decisions' that lead to the generation of a particular expert system, and this record is used as the basis for the explanation facility. These decisions, such as choosing not to develop a particular fact into a rule, or not implementing a certain function in the inference engine, are explicitly incorporated into the explanation facility for the system. They attempt to capture the knowledge brought to the surface during knowledge elicitation in a high-level representation from which the expert system's code may then be derived. This knowledge which is otherwise lost includes principles and facts of the problem domain as well as how these principles and facts are incorporated into the system, all key elements of an explanation facility. The developers of EES understood that by capturing the design rationale behind a system, they were making progress towards solving the explanation content problem, as well as the problems associated with both user and design documentation (Swartout, et al., 1991).

2.1.3 Aspects of Expert System Explanations

Gregor and Benbasat (1999) reviewed the base of empirical research into knowledge-based system explanation and developed a map of the constructs that are most prevalent in the field.

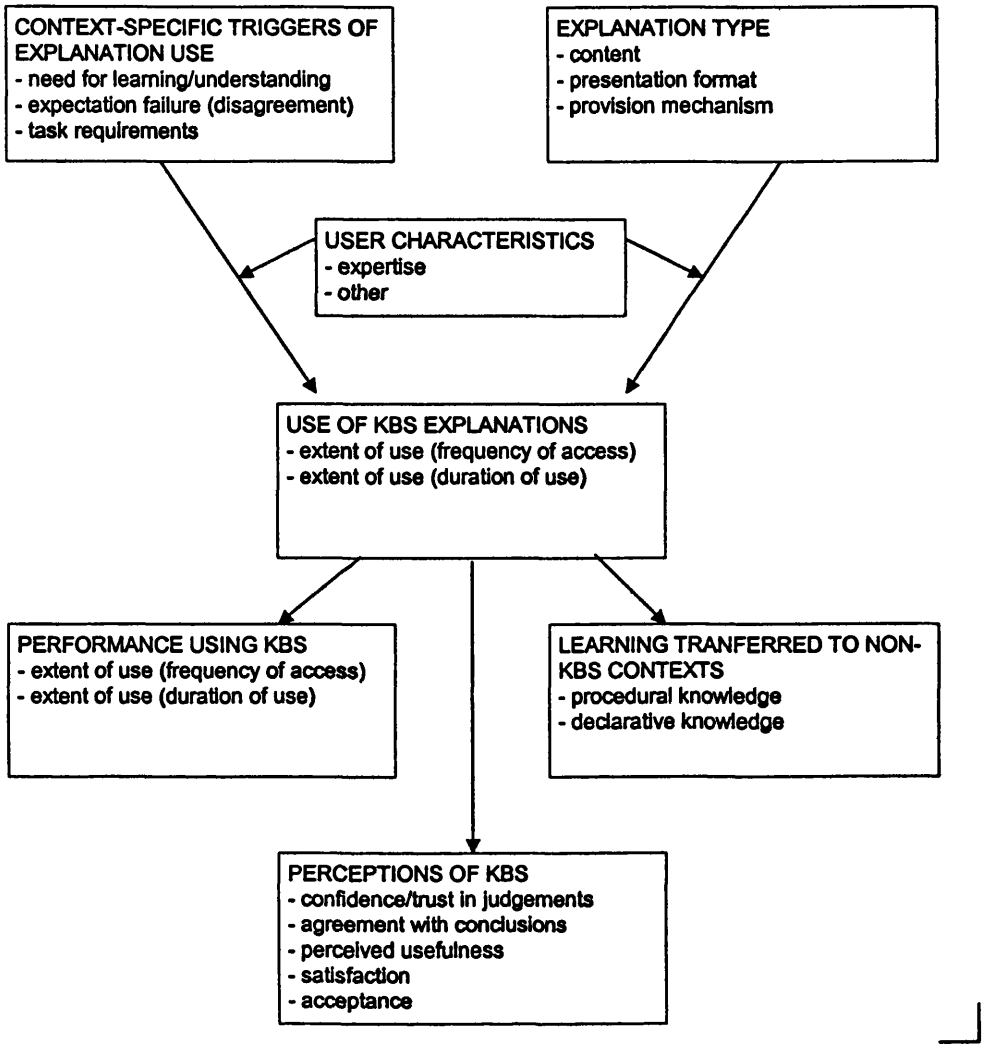


Figure 3 - KBS Explanation Constructs

Gregor and Benbasat use the constructs in the above diagram to derive a set of three research questions for the field of explanations in KBS including: do users of intelligent systems want explanations; do benefits arise from using explanations; and what types of explanations should be provided? Of particular interest here is their focus on the role of Toulmin's (1958) argument model as a framework for describing the content of KBS explanations. This approach to the representation of explanation knowledge was introduced in the system explanation literature elsewhere (Wick & Slagle, 1989; Ye & Johnson, 1995), and is discussed in depth in the later chapters of this dissertation.

According to Dhaliwal and Benbasat (1996), knowledge-based system explanations may be described by a three-part typology:

1. *Why* and *how* explanations as exemplified in foundational work such as the MYCIN project.

2. *Strategic* explanations derived from meta-knowledge of the problem-solving strategy employed by experts in the domain, for example, NEOMYCIN.

3. *What* explanations that provide object definitions and descriptions of the decision variables that are manipulated by the system.

This typology corresponds to the "epistemology" developed by Clancey (1983, 1986) and to other related work (e.g., Kent, 1978) where explanations of knowledge-based systems are made up of structure, strategy, and support knowledge. Indeed, the consensus view among expert systems researchers is that Clancey's framework best describes the different types of explanatory knowledge required for system explanations (Dhaliwal & Benbasat, 1996).

Another approach to expert system explanation is the critiquing model, in which the conclusions generated by an expert system are automatically reviewed as they are provided to the user (Miller, 1986; Silverman, 1992). The critiquing approach involves the user initiating a function while the system traces the function, at the same time developing a script to perform the same function using normative task plan knowledge that has been captured in the system development process. The system then compares the two approaches to performing the task, the user's and the system developers', and provides justifications for why one approach was superior to another (Langlotz & Shortliffe, 1989). The output from the critiquing module is then used as the basis for an explanation for why one task plan may be superior to another.

A shift that has occurred in KBS research has been from the development systems employing deep, but narrow and therefore brittle knowledge bases to ones that make use of more broad and shallow information (Swartout, 1996). This development was motivated by the recognition that KBS systems should support rather than supplant expert users as they carry out tasks in their domain. This approach acknowledges the role of humans in the problem solving process and the subservient but important role played by technologies that can help exploit prior knowledge.

Explanation Knowledge Capture & Representation

A useful point of entry into a discussion of knowledge capture and representation is the "Frame Problem" as described by Dennett (1990). The Frame Problem asks whether it is realistic to think that a knowledge engineer could identify and codify all of the knowledge necessary for a system (in Dennett's example, a robot) to function intelligently within a given domain, in particular, whether the knowledge engineer could predict all of the effects that might emerge when the knowledge that has been identified is put to use in the domain. The Frame Problem suggests that we can never adequately delimit the boundaries of the knowledge base needed to function in a domain. There would always be additional knowledge which escapes identification and codification but upon which our identified knowledge depends. Further, even if all knowledge elements could be identified, the difficulties inherent in choosing between those relevant to the immediate task are intractable.

The problems of computer-based explanation serve, unfortunately, as an excellent example of the effects of the Frame Problem on a knowledge-based system. Suppose that we wanted a system to explain completely, to a novice user, the functioning and rationale behind a transaction-processing module in a computerized double-entry accounting program. The user posts the transaction and then asks the system to explain what has occurred and why. The user knows only that he has just posted a transaction for a sale of goods for £100.00. What does the system 'know', and therefore what can it possibly explain about what it has done with the transaction? It may 'know' that it actually posted the following entries:

Debit		**Credit**
Accounts	Receivable	Sales
117.50		-100.00
		VAT
		-17.50

The user (naive in both computing and accounting) may reasonably ask, among other questions concerning this transaction, why the amount has been incremented from the £100.00 they entered to the £117.50 in the final result. The knowledge needed to explain this increment includes the fact that both the customer and the goods involved in the transaction were in taxable categories and the current rate of tax is 17.5%. Problems of the lack of explanatory power in the system appear when the user begins

to ask for explanations of, for example, the different rates of VAT: 17.5%, 0%, and Exempt, and how they are applied to the different customers and goods that may be involved in a transaction handled by the system. There is arguably no complete solution to the Frame Problem.

As the Frame Problem suggests, attaining a measure of completeness in the knowledge base is one of the most demanding problems in knowledge engineering. Cooke (1989) argues that completeness in knowledge engineering can be achieved through redundancy of information, it is desirable in an explanation facility to have a number of ways to explain the same concept. To Cooke, the first stage of knowledge elicitation is the identification and generation of relevant domain concepts. In his studies, the interview method of knowledge elicitation produced the most redundant explanation of the same concepts. Redundancy of ideas gathered in the knowledge elicitation process is a measure of the completeness of the ideas gathered. After the set of redundant concepts has been gathered, asking experts to organise the information into a conceptual hierarchy or tree is one way of revealing missing components in the knowledge base.

Another significant problem that knowledge engineering must overcome is the issue of atomisation or necessary and sufficient granularity of the information that is appropriate to the system. There is no single definition of granularity that provides any system with appropriate multi-level explanatory power. A method of overcoming the granularity problem is the binding of meta-rules with original source citations, perhaps stored in a database separate from the knowledge base (Bainbridge, 1988, p. 181). Meta-rules provide the user with a high-level, definitive explanation of the concept under discussion as well as a level of detail that would be inefficient if stored within the knowledge base itself. Other support knowledge for use in the explanation process includes storing causal or mathematical models and statistical studies related to the domain and made accessible to the system during the explanation process (Clancey, 1983). The developer may also choose to write their own expanded explanations of the algorithms, structures, and rules used in the system (Berry & Broadbent, 1987). These explanations are indexed to the part of the system to which they relate and are made available whenever this part of the system is queried for an explanation.

One problem with the use of meta-knowledge as described above is the additional development overhead that the system and the knowledge engineer must assume. In effect, the knowledge engineer is required to develop skills in "knowledge taxonomy" (Bainbridge, 1988; Clancey, 1983). Clancey proposes "typing" knowledge, much like data structures are typed in programming languages, according to a rigid classification system grounded in sound knowledge theories. There is no doubt that this process of abstraction and classification of meta-rules from a large and complex knowledge base is a difficult and time-consuming process. In later chapters I attempt to show how this activity can be made an integral part of the system development process with measurable benefits, not only to the user, but to the developers, that help to justify the effort.

Higa et al. (1992) approach the problem of knowledge base epistemology from the field of database/knowledge base coupling. They describe two types of knowledge: extensional (episodic) and intensional (semantic). Extensional knowledge is represented in a database by records or rows of data and intensional knowledge is represented in the structure of the database itself. Intensional knowledge is knowledge outside of the factual knowledge contained in a database. One form of intensional knowledge is structural knowledge that describes the structural dependencies and constraints among data. They present the Structured Object Model (SOM) which uses objects, attributes, and two types of relationships: aspects and specialisations to represent data semantics.

In general, the knowledge elicitation and knowledge engineering phases of expert system development projects have not been fully exploited as sources of the more complete domain knowledge needed to provide adequate explanations (Clancey, 1983; Silvestro, 1988; Berry & Broadbent, 1987; Dhaliwal & Tung, 2000). During these initial project phases, the knowledge engineer becomes deeply involved with the domain and domain experts in their attempts to elicit the central information structures and processes in a given body of expertise. Especially if the knowledge engineer or analyst is unfamiliar with the domain, it is at this time that the most complete understanding of the domain is made explicit, both in their minds and in those of the domain expert. Unfortunately, what normally happens is that this base of full and explicit knowledge is then re-condensed into production rules that fail to capture much of the control knowledge that actually drives the expert reasoning

approach. As highlighted by Clancey (1983), much of this knowledge is captured implicitly, such as in the order in which a set of rules will fire, but these representations are inaccessible to the IEF.

Only one study has been identified that explicitly proposed a distinct method for capturing explanation knowledge (Dhaliwal & Tung, 2000). This study involved using a group support system (GSS) combined with the Delphi method, an iterative approach to eliciting opinions and gaining consensus among multiple, typically anonymous parties (see Lindstone & Turroff, 1975). While this approach appears to have resulted in the development of high quality explanations, with the added benefit that a record of the process by which they were developed was captured in the GSS, Dhaliwal and Tung acknowledge that this approach is time consuming and costly.

The Content of Expert Systems Explanation

It has been argued that a fundamental problem in the history of expert system development is that the purpose of different system components has been "overloaded", that is, production rules and other structures were called upon to play a diverse set of roles from defining terms in the problem domain to controlling the problem solving strategy of the inference engine (Swartout, 1996). This conflation of different kinds of information, all with different roles, into a generic rule form has resulted in knowledge bases that were difficult to use, reuse, understand, and maintain. An approach proposed to alleviate this problem is the development of ontologies that provide a more rigorous definition of the structure and purpose of the information managed within a given system (Swartout, 1996).

Dhaliwal and Benbasat (1996) argue that there is no sound theoretical basis underlying the various classification schemes that have been applied to explanations in the IEF domain, though they acknowledge that Clancey's (1983) structure, strategy, support framework has emerged as the dominant model. In one attempt to define a generic ontology for KBS, Wenger (1987) classified the types of knowledge used in expert systems and their derivatives such as GUIDON and NEOMYCIN as general principles, common world realities, definitional and taxonomic relations, causal relations, and heuristic rules. Though a useful starting point, this classification scheme still appears too generic to guide elicitation, structuring, and delivery of explanatory content.

Another attempt is that of Swartout and Smoliar (1987), who define three kinds of knowledge necessary for explainable expert systems:

1. Terminological knowledge - the vocabulary used in the domain to describe its concepts and relationships.

2. Domain descriptive knowledge - specific knowledge required to solve problems in the domain

3. Problem solving knowledge - 'how to' knowledge that allows the knowledge to be applied to problem solving in the domain.

They argue that *terminology* is seldom included in the knowledge base, despite its importance to system builders as they acquire domain expertise. In their ontology, terminology is divided into two types: lexicon and definitions. The lexicon provides a set of symbols for representing the entities relevant in the domain. Definitions are used to justify a given entity's membership in a class defined by the lexicon. *Domain descriptive knowledge* can be causal knowledge, descriptive knowledge (in well understood domains), or probabilistic knowledge (in not very well understood domains). *Problem solving knowledge* is defined as a set of plans that have *capability descriptions*, which are the goals the plans can achieve. Plans also have an associated set of *methods*, which are the specific steps followed by a plan to achieve a defined goal.

Lamberti and Wallace (1990) have examined the factors that impact the efficacy of the user-interface to intelligent systems. One factor that became a focus of their research was knowledge presentation format; in particular, whether explanations of system behaviour that included procedural versus declarative knowledge were more effective. By procedural knowledge they refer to displays of the inference rules that fired in a given problem solving session. Declarative knowledge refers to static information about the system and the domain in which it is applied. They argue that both types of information are useful in the explanation equation. Declarative knowledge is useful because it can be decomposed into modular facts that are tractable and reusable in different contexts, and save us from redundant knowledge engineering efforts (Chandrasekaran, et al., 1999). Procedural knowledge helps the user to understand how a given fact or event relates to the context of use.

During expert system development, the details of a given problem solving strategy, often followed unconsciously by the human expert, are made explicit during knowledge elicitation. However, when this base of knowledge is then encoded into structured system rules, much of the strategy that has been brought to the surface is again made implicit during the representation of this knowledge in the rule base. Explicit representation of the problem solving strategy employed by an expert system is a prerequisite of an adequate explanation facility. Clancey (1983) argues for the use of a domain-independent representation of this strategy in order to provide more universal explanations. He describes the act of generalizing observed phenomena into distinct problem solving steps as a prerequisite to providing complete knowledge and process explanations. From specific instances of domain rules that are not understood by the naive user, this "generalization tree" may be ascended until a related, and presumably understood concept - expressed as a meta-rule - is found. A strategic explanation facility developed following this approach attempts to clarify the goals and plans of the current inference session by abstracting from the process a general form of problem solving and relating it to a specific act performed by a system within the current domain.

In later reflections on the seminal MYCIN, GUIDON, and NEOMYCIN projects, Clancey began to articulate a more 'holistic' perspective on the epistemology required to frame explanatory knowledge (Clancey, 1993a; Clancey, 1993b). In this work he began to conceive of the meta-rules identified in the earlier work as a sort of *argument* that could be used to describe the strategy behind a system's behaviour. Though in this later work he largely reaffirms his commitment to strategy, structure, support knowledge as the basic epistemology of expert systems explanations, one important change to the framework was to limit the scope of strategic knowledge to domain-general, from domain-independent (Clancey, 1993a).

According to Dhaliwal and Benbasat (1996), models of explanation must be developed within a particular domain of tasks and types of users. The content of an explanation model, the information with which it is populated, must be specific to a particular domain, to the user types, and to particular usage scenarios. These arguments have serious implications for efforts to develop models of system-generated explanation and ontologies of explanation content that can transcend any particular system use context.

2.2 Intelligent Tutoring Systems

Intelligent tutoring systems (ITS) are systems that are designed to deliver instructional material in a manner approximating that employed by a human tutor. A focal point in ITS research is the challenge inherent in providing individualised instruction using knowledge manipulated by a computer program (Angelides & Paul, 1993). 'Intelligence' in an ITS refers to the system's ability to reason about the differences between a hypothetical expert's domain knowledge model and the domain knowledge model possessed by the learner (Freedman, et al., 2000). ITS are used in a diverse set of domains including business games/simulations (Angelides & Paul, 1993; Siemer & Angelides, 1994), emergency room cardiac resuscitation (Woolf, 1996), and traditional fields of study such as mathematics (Al-Jumeily & Strickland, 1997). The literature of intelligent tutoring systems has been informed by research into IEFs and in turn, IEF research has been informed by research into ITS (Wenger, 1987). Since the idea of communicating knowledge using information models and procedures for manipulating them is central to the field of ITS, the nature of these systems is briefly reviewed in this section.

An ITS has four basic components: a domain model, a student model, a pedagogical model, and the system's user interface (Freedman, 2000; Sleeman & Brown, 1982). The first of these is a representation of an expert's domain knowledge, the second is the system's model of the learner's knowledge at a given point in time, the pedagogical model manages the sequence of learning activities designed to progress the student towards a grasp of the domain expert's model, and, finally, the user interface manages information display and input/output functions (Vasandani & Govindaraj, 1995). The interactions of these modules rely to a large degree on a predefined description or plan of the user's task along with a series of learning milestones that the user must achieve in order to advance through the material. ITS generally track the progress of a learner through a pre-defined set of learning activities corresponding to the domain model, with the goal of attempting to 'repair' the student model when it deviates beyond some threshold from the system's domain model. ITS are typically plan or scenario-based, meaning that the path or paths through a given set of domain knowledge is predefined by the designers of the system (Woolf, 1996).

Some have argued that the central element in intelligent tutoring system research is the role of experiential learning and that one of the key goals of an intelligent tutoring system is to transform factual knowledge into experience-based knowledge (Sleeman & Brown, 1982). A key tactic in this approach to pedagogy is the exploitation of a user's learning problems, their 'flounderings', which are transformed through interaction with the tutoring application into concrete, experience-based knowledge. An additional benefit of the ITS approach when combined with domain scenario simulations is that experiential education may be provided in domains where real-life flounderings might have catastrophic consequences, for example, in the control room of a nuclear power plant (Vasandani & Govindaraj, 1995).

Sleeman and Brown (1982) identify a number of significant challenges that have emerged from efforts to build tutoring systems. Among these are that baseline models of the instructional material begin at the wrong level of detail, either over or under estimating the student's knowledge. Learners may become frustrated as they work to understand the system's domain model while at the same time the system adapts instruction to its evolving student model (Browne, 1990). Another issue is that the tutoring strategies used in existing systems are often based on ad hoc principles of learning. They argue that the learning strategies embedded in these systems must be based on psychological theories of knowledge representation and belief revision. Finally, the limitations of human interface technology result in significant constraints on the learner's ability to communicate with these systems, thus not allowing the systems to fully diagnose the user's learning problems.

The putative separation of process and information in expert systems IEFs such as MYCIN and GUIDON was seen as a useful strategy to apply in the tutoring task since the knowledge managed by these systems is represented and stored in a modular format and therefore potentially shareable across applications and possibly across domains (Wenger, 1987). However, some ITS researchers argue that at least some of the pedagogical knowledge needed for effective instruction is domain dependent (Freedman, 2000). According to Vasandani and Govindaraj (1995), the efficacy of ITS in highly complex domains has yet to be demonstrated. They argue that in fields such as engineering design, the complex interactions between system components present significant challenges for the capture, organisation, and delivery of useful instructional material.

A fundamental difference exists between research into intelligent tutoring systems and other instructional technologies, and research into integrated explanation and help facilities. The *raison d'être* of an ITS is to convey information and promote understanding of some target topic. *Integrated* explanation facilities and help systems, on the other hand, exist as an *adjunct* to the system and application domain in which they are embedded. As discussed later in the thesis, this distinction becomes important in attempts to justify the investment required to engineer explanation knowledge and capture/delivery tools. Although an explanation facility can benefit greatly from tutoring system research, especially the problems of how to convey an understanding of the problem domain, much of the work is inappropriate for a facility which is concerned with explaining the system into which it is integrated. Whereas the motivation and role of a tutoring system is to convey to the user an understanding of domain, the explanation facility's primary goal is to facilitate the usability of the system into which it is integrated. Clancey (1983) points out that teaching how to do something is very different from explaining why something was done. While this may be the case when providing *how* explanations, it obviously is not when the *why*, the rationale for the problem solving approach, needs to be explained as well. Still, as the function of tutoring systems is generally to convey understanding and the seminal work on IEFs and tutoring systems has been symbiotic, they are discussed here briefly for their unique contributions to the field.

Nathan (1992) challenges the idea that a computerised tutoring system should guide the student through the learning process, claiming that such systems do too much of the thinking for a student and thus do not encourage them to develop their own skills. He argues instead for the use of what he calls "unintelligent" tutoring systems, systems designed to help students learn to learn. Nathan supports this argument by citing research that suggests that minimising students' memory work loads, making learning goals overt, providing the context for the instruction, and focusing on iterative problem solving are principles central to the development of tutoring systems. His work runs counter to much of the research in tutoring systems, which suggested that immediate feedback regarding student errors is critical to the instructional task. He highlights recent studies that focussed on the problems with attempting to monitor students closely, knowing all of the ways in which a student can err, and being able to diagnose the error that the student has committed. The

immediate feedback model is criticized for interrupting a student's thought process and memory trace with the result being impaired learning. These studies suggest that students learn most effectively when they can see the effect of their errors and must reason causally about the behaviour of the system. Nathan also suggests that delayed feedback is more effective because, instead of allowing the student to guess at the answer to a problem and then having the system immediately correct their mistakes, they must consider each of their actions more carefully since the cost of mistakes often accumulates until finally reported to the student.

Research into intelligent tutoring systems provides an additional perspective on the task of generating explanatory material in a software system. However, their reliance on the maintenance of complex domain knowledge models, student models, and a mapping of their differences poses significant knowledge engineering, computational, and human computer interface challenges. Given the number of domains where an ITS might be applied and the individual differences among its target user base, development of a generic system may prove an elusive goal (Wu, 1993).

2.3 User Modeling, Explanation, and HCI

Much of the research into computer-generated explanation has been directed at the problems encountered in attempting to manage the subtleties of explanatory dialogues (Maida & Deng, 1989; Cawsey, 1992; Moore, 1995). Many explanation facility researchers claim that the most critical element of system-generated explanations is the dialogue management approach employed (Lamberti & Wallace, 1990). Cawsey (1992) argues that in a long and complex explanation users should have the ability to interrupt the system to ask for clarification of the information being given. She claims that "verbal explanations of all lengths are intrinsically interactive". Conversely, the system should keep track of the user's level of understanding and adjust the information being given as appropriate. An integral part of explanation delivery is reasoning about how information that is communicated affects the knowledge states of the two or more parties participating in the communication (Maida & Deng, 1989).

Recent efforts to include IEFs in information systems have been characterised as efforts to improve human-computer interaction (HCI) and system usability (Dhaliwal and Benbasat, 1996). Explanation may be defined as the process of one actor

attempting to convey an understanding of a particular concept to another. According to Shank (1986), achieving this understanding is not a discrete, but a continuous process. It is not simply that one either understands or fails to understand a given concept. Levels of understanding exist on a continuum and thus can be used as a metric to describe the success or failure of an explanation process. A condensed version of this continuum and a part of Shank's measurement scale and associated nomenclature are illustrated in the figure below.

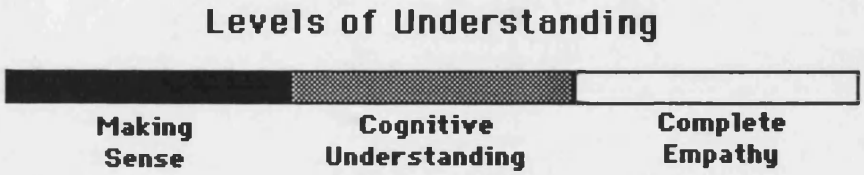


Figure 4 - Schank's Levels of Understanding

In Schank's model, the "Making Sense" level is described as minimal understanding, an elementary and perhaps incomplete grasp of the domain or concept under consideration. Shank considers this as the level of understanding that is conveyed between a human and a computer with currently available technologies. At the other end of the scale is "Complete Empathy", described as the level of understanding that may exist between two actors who have a rich shared set of experiences upon which to base the terminology and analogy of their explanations to each other, for example, two close siblings. "Cognitive Understanding" exists when it is possible to gain a fairly complete and accurate understanding of the domain given a reasonable amount and quality of communication.

One of the most significant problems with explanatory dialogues between humans and machines is the machine's inability to establish the background knowledge possessed by the human. In IEF research it is recognised that the ability to connect a listener's prior knowledge to the concepts in an explanation is crucial if we are to provide economical, usable explanations (Clancey, 1983). A key component of human-to-human explanation is that people are able to quickly establish at least a portion of this background knowledge and apply it to the construction of an explanation (Winograd & Flores, 1986). Antaki (1989) argues that the single most important aspect of an explanation given by an intelligent system is that it be believable. He suggests that in order to be believable, a system providing an explanation must know what the user

does not know, know what the user wants to know, know why the user wants to know it, and know what the user will accept as an explanation. Researchers have attempted to address these explanation dialogue issues through the use of user modelling techniques, as well as attempts to refine the dialogue management capabilities of explanation facilities to the extent that at least within a given explanatory dialogue, information that has already been provided to the user is taken into account when constructing the next explanation datum.

Dhaliwal and Benbasat (1996) explore the distinction between feed-forward, information provided to the user before execution of a task, and feedback, or outcome information provided after the task has been executed. There are two distinctions between these types of information, temporal order and case specificity. Since the task has not yet been performed and no outcome information is available, feed-forward information is by necessity of a general nature and can tell the user nothing about the exact impact of the task they are about to perform. On the other hand, feedback information has the advantage of having the potential to incorporate information specific to the instance of the operation that has just been performed. Dhaliwal and Benbasat characterise the distinction between feed-forward and feedback as corresponding to learning by being told versus learning by doing. They cite studies that show that feedback is more effective in fostering system learning and task performance but claim that overall research is inconclusive as to which method has the advantage of overall effectiveness.

In their studies involving the observation of experts assisting novices in the execution of complex tasks, Johnson and Johnson (1993) found that the flow of explanation generally consisted of experts making a determination as to the knowledge possessed by the novice, defining key terms related to the concept being explained, identifying and filling gaps in the novice's knowledge, and deciding when the explanation was complete. That explanation delivery might be formulaic suggests that planning may play a significant role in the construction of coherent explanations. According to Weiner (1989) the three target functions of planning in developing an explanation are what to say, how to say it, and how much must be inferred and how much can be assumed as principles that do not require systemic explanation. The latter problem in particular has proven particularly intractable as is discussed in the next section.

2.3.1 User Modeling

One of the most frequently cited problems with the development of computer-generated explanation is the human ability to determine at least a portion of the background knowledge possessed by the other humans involved in the explanation transfer (Winograd & Flores, 1986). Various degrees of empathy allow the participants in an explanation dialog to "ground" their statements in concepts and experiences familiar to all of parties involved in the transfer of knowledge. Winograd and Flores classify the various methods in this process of "grounding" as follows:

1. Experiential. When asked to justify a statement, the person explaining gives the steps that can be followed to achieve results conforming to the explanation given.

2. Formal. Rules such as mathematics or logic constrain the bounds of the domain and almost by default provide the explanation.

3. Social. Someone told me, I'm passing it on.

It is frequently argued that a system must be aware of the skill level and goals of the user currently using the system in order for a it to support a wide range of potential users while at the same time providing them with adequate explanations (Ellis, 1989). In particular, the requirement that different information be provided to different users depending on their skill level, from novice to expert, seems especially well justified (Swartout & Smoliar, 1987). One way to achieve the necessary user awareness may be by constructing a model for each of the different classes of user and determining at the beginning of a session which of these models best fits the current user. Allen (1990) considers the central task of these user models to be the prediction of users' behaviour through knowledge of their plans and goals.

The simplest form of user modelling involves users classifying themselves, for example as a domain expert, system expert or naive user, at the start of a usage session. Alternatively, they may be asked a small number of simple, preliminary questions while logging on to the system thereby allowing the system to automatically classify them. A similar approach involves users designating their preference for the use of a particular domain model in explanations (Suther, 1993). Preferences might include providing term definitions, when available, or supplying a minimal amount of

information for any given explanation request. Other types of user models include static and dynamic models where the former classifies a user once and the latter supports the evolution of the user model as the user becomes more experienced and competent in the use of the system. The latter, evolutionary type of user modelling is required for systems that are designed for instruction as they must adapt the material presented as the user progresses.

According to Berry & Broadbent (1987), the following components are the essential elements of a system with user modelling capabilities:

1. A usage log for each user.
2. Factors representing how much each user knows about the system domain.
3. Factors representing how much each user knows about the system.
4. The results of past usage sessions.
5. Interaction preferences.

A number of proposals have been made regarding the ways in which user models can be used to affect human-computer interaction. One such approach is based on a technique where an "overlay" of user knowledge is compared to that of a putative domain and system expert. Such an overlay, composed much like a polar graph in that variances between the layers are made readily apparent, provides a clear indication of the areas in which the user's skill level differs from that of the expert. The goal of the interface software utilizing such an overlay model is to minimise the overlay area by adapting the system to account for the gaps between the current user and the model expert.

There are a number of significant problems with the use of user models in information system interface software. Huber (1983) argues that because of the complexity of any efficacious user modelling technique, development and maintenance costs would outweigh any derived benefits. One problem reported by researchers is a bias by users against the idea that a system can classify them and then, based on this classification, retrieve and present a selection of information tailored to a fundamentally simple model of their abilities and goals (Allen, 1990). To achieve anything more than a rudimentary model of the user requires potentially intrusive psychometric testing to

derive personality or cognitive traits through the narrow medium of the user's interaction with the computer (Browne, 1990, p. 199). Assuming such metrics could be captured, the implications for the user's privacy raise a range of issues that complicate the system development process.

Among the other problems with effective user modelling is the phenomenon of 'hunting', a potential problem for any adaptive system (Browne, 1990). Hunting occurs when the system is in the process of developing a user model and the user is in the process of developing a system model. Both system and user can become confused (and the user very anxious and frustrated), as they try to hit these moving targets. Another is the user's perceived loss of control in trying to deal with an adaptive system as control of the system is critical to the user experience when using software tools to perform tasks (Stylianou, et al., 1992; Gill, 1996).

According to Foltz and Dumais (1992), effective information retrieval systems are largely dependent on a model of what the user is interested in, not in 'deep', psychometric data. This may not be relevant in the majority of information system applications as the user is assumed to be interested in the system for the simple reason that they require it in their work. Perhaps more important in the majority of information system applications is to decide which explanation of a particular issue or component is of interest to the user depending on their role in the organisation and their corresponding goals as defined in an organisational model including detailed job and task descriptions.

Among the more simplistic and pragmatic approaches to user modelling is that proposed by Kotterman and Kumar (1989) in their "user cube" model. The user cube defines end users as engaging with a system in one of only three ways:

1. Operation - engaged in the monitoring of the system
2. Development - engaged in the production of systems
3. Control - engaged in the decision-making of computing

Their taxonomy does not require the deep user knowledge entailed by many of the psychology-based user modelling techniques, thereby avoiding problems associated with perceived intrusiveness and individual privacy. In fact, with the user cube model,

users can be classified based solely on the level of security implied by their system access level. The user cube provides a potentially useful approach to tailoring explanations as it is concerned with the organisational role of the user, not their psychology. Information related to organisational role might be sufficient to provide a realistic framework for tailoring explanations if these role designations can be used to relate explanation requests to system tasks and usage scenarios.

Despite these issues, recent work on user interface designs that employ user models based on Bayesian networks have achieved some successes and wide exposure in the IS marketplace due to their inclusion as part of the Microsoft Office help system (Horvitz, et al., 1998; Hedberg, 1998). The Microsoft Office Assistant uses Bayesian models developed from protocol analyses of application users to determine the probabilities of areas where the user may need assistance combined with the probability that the user would be disrupted if offered program assistance. One of the insights gained from this work is that closer integration is required between system developers, who develop the system events used to prompt user model tuning, and system documenters, who attempt to provide the precise information required by the user based on sequences of system events that might suggest assistance is required.

Dhaliwal and Benbasat (1996) describe the two contexts in which knowledge-based system explanation facilities are used as the instructional context, i.e., for training, and what they call the *working* context, when the system is used in the process of performing some work-related task. The latter scenario introduces the learning versus working conflict (Carroll and McKendree, 1987). This conflict emerges when users are unwilling to sacrifice time to longer-term learning if it affects their short-term work performance. It is argued that explanation facilities must be integrated into the systems and work processes that they support so that users are able to improve their work performance using the explanation facility without making the explicit time sacrifices entailed by instructional modules that are separate from the system that they support.

To address the learning versus working conflict, Dhaliwal and Benbasat argue for the application of the cognitive feedback model (Todd & Hammond, 1965). This model is an attempt to address the question of when best to provide system users with

information related to the task domain. The cognitive feedback model is a derivative of the Brunswik Lens Model (Brunswik, 1952), which takes the following form.

$r_a = Fn\ \{G, R_e, R_s\}$

where

r_a – *achievement*

R_e – *predictive ability of the cue set provided, i.e., information about how cues in the set relate to the outcome of the system operation*

R_s – *predictability or consistency of the individual in terms of how they apply the cue set*

G – accuracy of the cue weighting, i.e., how well does the individual decision maker (user) use the cues in relation to the normative model

Dhaliwal and Benbasat argue that this model provides guidelines for the types of the information that might be provided in an explanation. It is applied to an expert system usage scenario as follows:

1. *Relationships between cues and criterion events in the task system (R_e)*
2. *Relationships between cues and criterion events as perceived by the user or cognitive system (R_s)*
3. *Relationship between the task system and the cognitive system (1 and 2) which they call functional validity information (G)*

According to their analysis of studies that have attempted to compare the value of these three elements in a system usage scenario, information about the relationships between cues and criterion events is shown to be the most effective. One study that investigated the role of expert systems and their explanation facilities in supporting the development of more accurate cue sets and cue weighting found that these systems did appear to improve performance in financial decision making tasks (Nah & Benbasat, 1997). Information about the relationships between cues and criterion events as perceived by the user, has been shown as not very effective in supporting task achievement. They argue that these studies show little support for the idea of user modelling as a crucial component of explanation facilities. Little work has been done on number three, information about the relationship between the task system and the

cognitive system, so there is no clear indication of its value. The different ways in which explanatory cues can be combined in a given scenario points to the role of creativity as one of the primary components of the human explanation facility (Shank, 1986).

Draper (1998) argues that use of instructional materials in the work context is an integral part of many professions. He offers as an extreme example that of librarians, whose speciality is not knowing all of the information in the library, but in knowing how to find it. In complex domains and when using complex tools, users cannot avoid making reference to instructional materials and other aids. Much of the current work into situated cognition suggests that these aids are inextricably bound up with performance of a task and that well designed aids and tools in essence become part of the task performers cognitive apparatus (Wells, 1998; Hutchins, 1995).

2.4 User Manuals and Online Help

The instructional materials provided with computer hardware and software applications are notorious for being difficult to use and are often considered a last resort when attempting to complete a task (Randall & Pedersen, 1998). The forms that instructional materials for IS have most commonly taken include hard-copy documentation, online help systems, and various forms of interactive and multimedia training materials including tutorials delivered on CD-ROM. Though studies suggest that training materials do increase user self-efficacy (Torkzadeh, et al., 1999), there is little consensus on what information users really need to use technology effectively (Roesler & McLellan, 1995). For expert users in complex domains, the challenge to technical communicators is to successfully mediate between these users, who already know how to operate in a problem domain, and a system that may represent a significantly different model of how tasks are performed (Hackos, 1998).

Until recently, the techniques used to structure both traditional print and electronic materials have drawn on the same theoretical models of instructional design. Though the field of computer documentation has changed little since its inception (Zachry, 1999), the development and widespread adoption of enabling technologies such as hypertext, the world-wide web, and enhanced search capabilities provide an environment with the potential to support a range of innovations in this area. This

section explores the evolving tradition of providing users with instructional materials for the IS that they use, with a particular focus on software applications.

It is often claimed that excellence in system design can result in systems that require no supplemental instructional materials (Draper, 1998). While this claim seems dubious in the case of complex systems in complex domains, a well designed human interface can help minimise the amount of technical documentation required (Hackos, 1998). A number of researchers have argued that users often ignore the information given in system documentation in favour of the 'illusion' presented by the user-interface (Zimmer, 1989; Pollard and Crozier, 1989). They argue that system users will often discard information that does not match or is contrary to the illusion that they have developed. Zimmer's solution is the use of interface 'repair mechanisms' to adjust this 'illusion' or system model to more faithfully represent the system context. At the simplest level, repair mechanisms are error messages from the computer or computer dialogs presented in response to requests for help from the user. The ability to identify exactly where the user's illusion deviates from the actual system model, where the error occurred or what exactly the user needs help with is critical to the effectiveness of these repair mechanisms.

Until recently, the development and evolution of online help systems has been constrained by the lack of any major paradigm shift that would support taking full advantage of the computing power now available on the average desktop. According to Turk and Nichols (1996), this lack of progress is largely due to the perception by corporate management that exploiting new online help and documentation technologies is largely a matter of converting text designed for print production to its online, electronic analogue. They argue that this state of affairs has limited developments in the field to the use of hypertext and increasingly advanced search facilities for large amount of help text available online. As users become more demanding, online help systems and integrated software wizards will be called upon to deliver more functionality such as data input validation, collaborative filtering in information retrieval tasks, and software task coaching (Patrick & McGurgan, 1993).

Roesler and McLellan (1995), developed a taxonomy of end user help content derived from the theoretical and empirical literature as well as their own end user studies. The taxonomy they constructed includes information defined as:

1. *Need-to-know-first* – basic information needed to begin using the application.

2. *What is* – definitions and descriptions of system objects and behaviours.

3. *What next* – information on how system enabled tasks are sequenced.

4. *How do I* – descriptions of how application-supported tasks are carried out.

One approach to defining the role of documentation in the milieu of computer use is as part of an *ecology,* which includes both painstakingly designed artefacts such as software and software manuals, and *ad hoc* supplements such as 'cheat sheets' and Post-It notes created and employed by end users (Zachry, 1999). A growing body of researchers are arguing that questions concerning the usability of an IS, its online help system, and its documentation are meaningless unless they account for wide range of factors that are present in these ecologies (Spinuzzi, 1999).

2.4.1 The Systems Approach

Historically, IS users have received IS instructional materials in the form of a thick, printed tome called a User Manual or User Reference. These materials were typically based on an approach to instructional design sometimes called the *system* or *software* approach (Carroll, 1990). Instructional materials that follow this approach are organised around the components of the system and represent a decomposition of the system's features, for example, windows, fields, menus, and buttons, with an emphasis on a thorough understanding of each of these individual components (Carroll, 1998; van der Meij, 1992). For each system component, declarative or concept-oriented information is provided to describe the purpose the component plays in the application domain. This focus on principles may be considered analogous to traditional, teacher-led instruction where a top-down, deductive approach is applied to the learning task (van der Meij, 1992). In recent years, comprehensive user manuals have been supplemented with reference cards and quick-start guides, which provide condensed information designed to help users become productive with a system more quickly (Patrick & McGurgan, 1993).

The system approach to documentation has been criticized for a number of usability shortcomings. Among these is the claim that in order to use this type of documentation, the user must first understand how the software works (Rettig, 1991). In other words, if the documentation is organised by system component name, users need to know which components relate to the action they desire to take. Another issue is that the system approach has traditionally emphasised comprehensive content at the expense of accessibility (Draper 1998). The fact that information exists and has been captured is of limited relevance if the costs of finding it outweigh the advantages of using it.

2.4.2 The Task-oriented Approach

The task-oriented approach to instructional design takes a strictly pragmatic view of the system documentation task. According to adherents of this approach, the documentation provided with a system should provide specific, step-by-step instructions for how tasks in the system's target domain are carried out. Brockmann (1990) describes task-oriented documentation as consisting of the following five basic elements:

1. Who performs the task?
2. What action begins each task?
3. What are the specific steps involved in performing the task?
4. What action ends each task?
5. Are there variations in hardware or in the general environment in which the task takes place that would alter how it is performed?

A key idea behind task-oriented documentation is that it shifts the focus of the instructional material from the perspective of the system developers and documenters to that of the users who use the system to perform particular tasks. A problem that arises with this approach is that in order to document a given task or usage scenario, analysts must be able to determine all of these scenarios in advance. The advent of event-driven and component-based software, where system components and actions are designed to be encapsulated and relatively independent of each other, means that task combinatorics are nearly impossible to predict in all but the most simple software

systems. Task-oriented documentation is also unable to predict the specific contexts in which even pre-defined tasks are performed and these contexts often introduce factors that might alter or conflict with the cookbook approach provided in the documentation (Rettig, 1991).

Hackos (1998) has found that in complex domains, expert users are often dissatisfied with task-oriented documentation. Instead of step-by-step instructions, these users require concise, conceptual information that will help them to understand how the systems designers envisaged tasks, tasks that these users already know a great deal about. Of particular importance to these users is for them to ascertain that the system designers really understood the domain and problems faced by those working within it. This information is essential for these users to trust the processes and output of the systems they employ.

2.4.3 The Minimalist Approach

Minimalism is an approach to computer system documentation and instructional design that is based upon the idea that users of systems look upon system documentation as a barrier to getting work done and that the job of the documentation professional is to minimise these barriers while still providing the 'minimum' information needed to use the software (Carroll, 1990). Reduced verbiage is only one of the success factors identified by this approach. Its ethos is that system documentation should be designed in such a way as to facilitate and support exploratory learning, and the minimalist documentation that is produced should be tested repeatedly to hone its efficacy in typical usage scenarios. The work that introduced the approach (Carroll, 1990) was based on a number of case studies in which minimalism led to the development of more usable and effective documentation, even in cases where systems manuals judged to be of high quality were already in use. The criteria by which the minimal manuals were tested included the amount of time spent actually reading the manual, the ability to perform a requested task, the speed at which the task was performed, and the number of errors made when attempting a task. Attempts at replicating Carroll's results have also been successful (van der Meij, 1992).

The minimalist approach, while explicitly resisting the prescription of specific instructions that might contradict its own philosophy, does suggest the application of

some general principles that result in more usable instructional materials. These guidelines include (from Brockmann, 1990):

1. Remove non-essential features from manuals, for example, introductions and summaries.
2. Focus on what users can apply immediately to their work.
3. Test the documentation repeatedly with users.
4. Provide screen shots and other graphics to allow users to synchronise what they are reading with what they are seeing on the screen.
5. Attempt to link new information to information already possessed by the reader.
6. Promote exploration of the system by intentionally omitting information from the documentation.

According to Brockmann (1990), minimalism is the evolution of task-oriented instructional design and its development was motivated in part by the higher costs associated with producing documentation that attempted to account for all of the different scenarios that might be encountered by a system user. A second central motivation for the minimalist approach are the principles of the 'production bias' and the 'motivational bias' that severely limit the effort that users of a system will spend working though instructional materials (Carroll & Rosson, 1987).

What the minimalist approach suggests in essence is that 'less is more' when it comes to system documentation (Carroll, 1998). Of particular importance to the approach is the role of iterative user testing, as this helps to identify those areas of a system that are most worthy of the limited number of words afforded by the approach (van der Meij, 1992). Users of minimalist documentation have reported feelings of anxiety over their lack of available information, especially in cases where the system they are using malfunctions (Draper, 1998). User testing prior to distribution of a minimalist documentation set can help to identify areas where these breakdowns occur and where the documentation might be expanded.

A key tenet of the minimalist approach is to promote exploratory learning by the system user. Wenger (1987) has highlighted the need to nurture motivated learners by ensuring they benefit from the satisfaction of independent knowledge discovery. Another benefit attributed to the minimalist approach is that, by promoting (and to a slight degree directing) exploratory learning on the part of the user, it assists the user in developing a sound mental model of the system model as instantiated by the systems designers (van der Meij, 1992). One study found that this effect is enhanced in *co-discovery* scenarios where two or more learners explore a system while verbally constructing a mental model of its operation (Lim, et al., 1997). However, as Draper (1998) points out, in a very complex system it may be the case that users require at least some basic instruction before they are competent to explore more advanced system features. In safety critical domains, exploratory learning might only be achieved if a simulated environment was available to allow harmless experimentation.

Brockmann (1990) argues that in addition to the usage benefits that appear to accrue from the use of the approach, minimalism also corrects the political error that is embedded in the Taylorist, task-oriented approach to documentation. Instead of being constrained and even de-skilled by a task-based approach, minimalism empowers the end user to actively, and therefore, it is argued, more effectively learn. In addition, the focus on iterative use testing during the development of the documentation effectively results in a participatory design approach to the development of documentation.

Among the arguments against minimalism are that experiments in support of the approach have typically used outmoded, systems-oriented documentation as the control (Farkas & Williams, 1990). These critics contend that as a control, modern, mixed-mode documentation sets that include a minimalist-like quick start guide would result in more realistic studies. Another concern that has been expressed is that as a method, minimalism has not been sufficiently operationalised to an extent that practitioners can easily apply it on real documentation projects (van der Meij, 1992). However, Draper (1998) found that despite the theoretical ambiguity of the minimalist principles, the approach has been relatively easy to transfer to practice and the results of applying them consistently positive.

Finally, a significant unanswered question is whether the minimalist approach will scale to larger and more sophisticated software applications (Carroll, 1998). Most of

the original base of minimalist research used a word processor or applications at a similar level of complexity when testing the efficacy of this type of documentation. Some recent work applying minimalism to complex systems suggests that the exploratory learning approach is preferred by expert users, but that they require access to deep conceptual knowledge in order to understand how a system operates in a domain (Hackos, 1998; Mirel, 1998). Carroll acknowledges that the scalability of the minimalist approach is a challenge, but suggests that providing information to support the development of "meta-skills", which in turn support the user engaged in exploratory learning, is a possible answer to this challenge. He hints at the role of design rationale as contributing to skill development on such systems, suggesting that access to design rationale would support the motivated and skilled user who, acting as a 'naïve scientist', attempts to build their own theories of how a complex system is put together.

2.5 Discussion

Expert systems are seen to partially embody human expertise in well-bounded, protocol-oriented domains. An important aspect of human expertise is the ability to explain and justify advice and conclusions. Though much of the research into expert systems has focused on the solutions that are the final product of their inference mechanisms, others argue that the process by which they arrive at these solutions may be more important to the usability of these systems (Whitley, 1990). In the seminal work on MYCIN and its derivatives, Clancey (1983) found that explanations of expert system behaviour relied on the structural, strategic, and support knowledge captured in a rule base, but that the linkages between these levels, an essential element of system functionality, were not explicitly represented and therefore unavailable to the programs attempting to construct complete and natural explanations.

Any system that makes use of a knowledge classification scheme, an epistemology in Clancey's view, must include explicit mechanisms that relate these knowledge types to the problem domain, to each other, and to the operation of the system in which they are used. Explanations break down without this mechanism, as they are unable to provide a complete justification chain to show why the system performed in a particular way given a particular use context. A use context is made up of the domain issue being addressed (a set of symptoms in MYCIN), the domain facts represented in

the system that are being applied to the issue (in MYCIN, production rules), and the manner in which a particular outcome is obtained by the systems operations (in MYCIN, the inference engine).

Of special importance to explanation is the relationship between strategic knowledge, the problem solving steps and heuristics for applying them, and structural knowledge, represented by the composition of the rule base. Access to such meta-knowledge provides the IEF with information that is abstracted from any particular rule or process, enabling it to make multiple, more efficient use of the meta-rules it contains (Bainbridge, 1988). Chandrasekaran and Mittal (1983) argue that the process of becoming an expert involves the compilation of generalised, condensed knowledge structures that are drawn from the larger body of domain knowledge. The knowledge structures that these researchers describe are analogous to the concept of meta-rules. Making this relationship explicit, and therefore available to the IEF, is a necessary first step in providing explanatory continuity within a given system. Clancey (1993a) argues that the support knowledge described in his classic 'Epistemology' paper would today be called a design rationale. He conceives of this support knowledge as that which describes:

> *"...the causal or social context that justifies a rule, an objective documentation for why a rule is correct."*

Methods for representing structure-strategy-support relationships were not explicated in the MYCIN, Guidon, and NeoMYCIN projects and the role of support knowledge beyond the essential insight given in the quote above was not developed further. The model below provides one way to conceptualise these relationships.

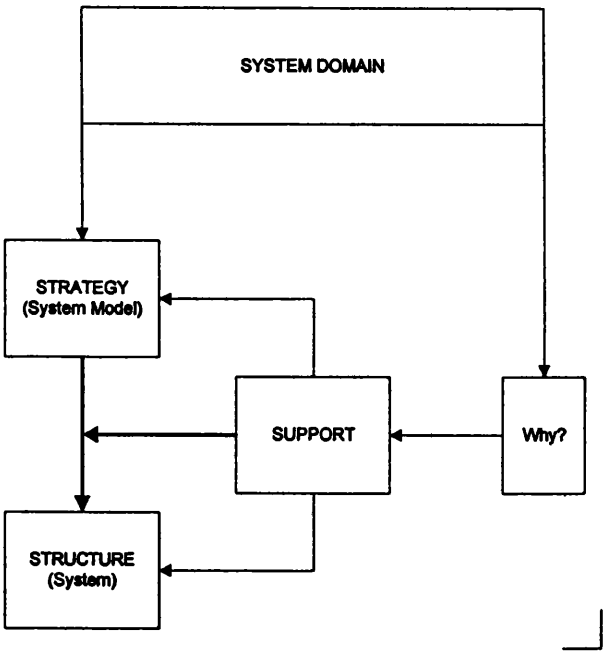


Figure 5 - System Structure, Strategy, Support Explanations

The model suggests that support knowledge plays a central role in the translation of domain knowledge into a system model and of the system model into a system structure. Why explanations provide the *justification* for how the system model is formed relative to the problem domain, and in how the translation from system model to system structure is performed. The next chapter will explore the nature of these why explanations further and provide a framework for the kind of information they employ in the IS domain.

The EES project team also recognised the essential role of the linkages between different levels and types of system knowledge when constructing IEFs. They proposed a separation of the IEF from the main body of the expert system program and a mechanism based on modular *capability descriptions* that link system goals to the plans a system uses to achieve those goals. Through the processes of specialisation and reformulation, the EES IEF is able to relate a given system operation to some directly relevant high-level strategy (through specialisation) and to other analogous, potentially explanatory strategies employed elsewhere in the system (through reformulation). This mechanism to relate meta-information to specific system and domain facts in the development of a system explanation is a key insight from the EES project.

By recording the design 'decisions' made by the EES automatic program writer and by including these decisions in the knowledge base used by the IEF, the EES team

explicitly acknowledged the role of *design rationale* in system explanations (Swartout, 1983; Swartout, et al., 1991). However, in expert systems, much of the processing implemented in the inference engine, as well as the structure of the production rules they use, are relatively homogeneous. This is not true in most IS where systems might be called upon to provide a wide range of services employing a host of different data structures. This means that models employed in systems such as EES, which can produce a design specification, or rationale, automatically based on a predefined domain specification, may not apply to more generic software systems. However, the insight from EES that domain information is used to drive specification, and that this grounding of the specification in domain information forms the basis for richer models of program explanation, is a foundation of the work reported in this dissertation.

Neither MYCIN nor EES addressed in any depth the 'real world' questions regarding where explanatory information comes from, how it is obtained, and the incremental costs added to a system development project that attempts to capture this information. One approach involving the use of group support systems combined with the Delphi method was presented (Dhaliwal & Tung, 2000), but the incremental cost of a distinct explanation knowledge capture phase seems prohibitive for most IS development projects. To the greatest extent possible, explanatory knowledge elicitation, capture, and representation should be a natural by-product of the systems analysis and design processes. Given the high cost of IS development projects and the time-to-market pressures that increasingly weigh upon project sponsors and their development teams (Cusumano & Selby, 1997), it is important that proposed methods for capturing and using explanation knowledge be developed with regard to these factors. One such method is proposed in Chapter 4 and investigated in later chapters.

Theories in the field of artificial intelligence are sometimes divided into two broad categories: mechanism theories and content theories (Chandrasekaran, et al., 1999). Content theories help us to understand the entities, their attributes, and their relations to each other that are the focus of our research efforts. In order to successfully capture, represent, and use explanatory knowledge in an IS, a clearly defined knowledge classification approach is required with the knowledge describing the taxonomy made explicit (Swartout, 1996). Taxonomies such as these should be grounded in theory so that they can be tested empirically, used in practice, and

evolved (Dhaliwal and Benbasat, 1996). It is important to recognise however that the knowledge engineering process necessarily involves both systems and domain specialists working together to achieve consensus on what constitutes the most important elements of a knowledge domain. Given the different perspectives and often competing goals of those involved in a given project within the organisational and social context, such consensus may prove to be elusive (Whitley, 1990). The work presented in this dissertation is an attempt to develop a content theory of IS explanations that acknowledges the complex context in which IS development projects occur.

Research into intelligent tutoring systems (ITS) has highlighted the importance of a student or user model in the delivery of instructional, explanatory material. However, attempts to develop software and data structures in support of such models have encountered difficulties in accurately mapping the development of student knowledge as they work with these systems. Human-to-human explanations are intrinsically interactive (Cawsey, 1992). The efficiency of human-to-human interaction, which makes use of 'channels' so far unavailable in computer-to-human explanation such as gesture, has proven extraordinarily difficult to reproduce in software systems. However, simplistic approaches such as stereotyping the user according to a fixed set of skill or knowledge levels, and very advanced Bayesian models of user activity, such as those developed at Microsoft for use in the Office Assistant, have demonstrated progress towards the goal of more naturalistic computer-to-human explanations.

The related fields of computer system documentation and online help systems are evolving rapidly way from the system view of instructional design, where material is organised based on the structure of the system and each system component is comprehensively documented, to a so-called minimalist model designed to encourage exploratory, experiential learning. Though the minimalist approach may be read to suggest that explanations of system concepts are just one more barrier to user understanding, little evidence exists as yet to support the minimalist approach for instances of sophisticated systems in complex domains (Draper, 1998).

The task-based and minimalist approaches to developing explanations of IS functionality both suggest that these explanations should provide information and

examples to relate system features to users' work (or play). Clearly the most prevalent and important tasks in a system domain should be explained with examples. However, efforts to create explanations that take into account all of the possible sequences a user might put together to deal with emergent, situated use contexts are likely to be unsuccessful (Suchman, 1987). It is argued instead that a modular approach to the design of explanation structures must be achieved, one that supports system users as they combine system features in unpredictable ways and ensures that potentially portable system components can be migrated and recombined while retaining their underlying explanations.

Finally, an important insight from the movement towards minimalist approaches to providing system information is the importance of promoting directed exploration through the materials provided (Carroll, 1990). Explanatory materials should be delivered in an easy-to-navigate format and users should be encouraged to explore the knowledge base when information needs arise. The system's inability to determine a users precise information needs and the potential intrusiveness of unprompted explanation suggests an approach to information delivery that in essence allows system users to construct their own explanations from the knowledge base provided. This approach is adopted and developed later in this dissertation.

3 Theories of Explanation

This chapter reviews the most important theories from philosophical investigations into the nature of explanation (Haynes, 2000). From each of these theories is derived a concept or group of concepts that is regarded as its essence, the central contribution to the debate on the nature of explanation. The evolving IS explanation framework draws freely from these theories with the additional constraint, pointed out by McCarthy (1995), that many philosophical theories, though rigorous, are not sufficiently well defined to be implemented in a computer program. These concepts are then related to both the literature on IEFs covered in the previous chapter and to the more general context of information systems development and use. A framework for what constitutes an explanation of IS is derived from this integration and this framework is then discussed. An approach to obtaining the information needed to populate this framework is proposed in the next chapter. Subsequent chapters investigate the efficacy of this approach relative to the framework described here.

As discussed in the last chapter, much of the IEF research to date has relied upon definitions of explanation drawn from a dictionary of English (e.g., Gregor & Benbasat, 1999; Ye & Johnson, 1995; Eberhart, 1995). Here are the three drawn from *The New Shorter Oxford English Dictionary*:

1. *The action or an act of explaining.*
2. *A statement, circumstance, etc., which makes clear or accounts for something.*
3. *A declaration made with a view to mutual understanding and reconciliation.*

These definitions, while intuitively appealing, appear more useful as explanation goals rather than as rigorous descriptions of the structure and content required of an explanatory statement. Another conception drawn from the existing IS literature is that whatever answers a what, how, or why question constitutes an explanation (e.g., Dhaliwal & Benbasat, 1996). However, as Draper (1988) shows, what, how, and why questions are typically interchangeable: What made you so sad? How come you're so

sad? Why are you so sad? Clancey's (1983) structure, strategy, support model was a first step towards the development of a more detailed explication of system explanations and some more recent efforts have begun to develop an account of the theoretical foundations of explanation based on an analysis and integration of empirical research in the field (Dhaliwal & Benbasat, 1996; Gregor & Benbasat, 1999).

Theories of explanation are at the heart of debates in the philosophy of science and there exists a vast body of work in the philosophy of science that addresses issues related to the structure, content, and purpose of explanations. For collections of the essential readings see Ruben (1993) and Pitt (1988); for a concise overview see Ruben (1998). This chapter is not an account of the metaphysics of explanation in the philosophy of information systems science, nor does it represent a theory choice where one or the other is seen to best apply to the explanation of IS. This chapter is an attempt to ground the notion of explanation of information systems in the relevant philosophical literature, to develop some theoretical anchors based on ideas from that literature, and to extract those concepts seen as most relevant for use in a theoretical framework for explanations of information systems.

3.1 Explanations & System Design

Kroes (1998) argues that technological artefacts embody both a physical and a functional essence. Although the physical structure of a technological artefact is bounded by physical laws and therefore may be at least partially described through the application of nomothetic explanations (discussed below), these artefacts are designed by humans for a purpose or function and therefore are amenable to functional explanations within the context of their use. Though Kroes makes a clear distinction between the "black box" physical structure and the functional purpose of an artefact, it is not clear that this line may be so easily drawn. It is impossible to separate the function of an artefact, for example, a software system, from the constraints that impacted its design and still provide a comprehensive explanation of why it behaves the way it does.

Software systems may be very complex and still not be bound by strict physical constraints, for example, processing speed, storage, or network bandwidth. These

systems can be designed and built to solve a given domain problem in a seemingly infinite number of ways. Kroes calls this the "underdetermination of designs by requirements". In addition, for organisational systems the scope of this underdetermination is extended by the ambiguities that arise when attempting to clarify a single set of requirements based on input from project stakeholders with a potentially vast array of perspectives (Checkland, 1981). Still, the number of ways that a software system may be practically implemented is closely bounded by logical, cognitive, communicative, end user, and organisational constraints as well as the regulatory, ethical, and professional constraints that may affect a design in a given domain. As will be shown in the rest of this chapter, physical and logical laws, standards and norms of various kinds, models and abstractions, functional requirements and different perspectives on these requirements are all combined in the design process where choices are made as to how they will be translated into a system architecture, and all play a role in explaining the resulting artefact.

3.2 Why Look at Formal Theories?

An essential point must be made about the distinction between theories of explanation in the philosophy of science and what counts as an adequate explanation in a practical, operationalized context. A continuum may be described between accounts of 'everyday' explanation given in psychology (e.g., Antaki, 1988; 1989; 1994) and philosophical accounts of explanation. Whereas the subtleties of everyday explanations between people are so complex that we may never achieve the required level of implicit knowledge recognition in a computer-based system, the philosophy of science is 'merely' concerned with attempts to describe the structure of a complete and ideal explanation (Draper, 1988). Though as an epistemic concept theories of explanation are examined most closely by philosophers of science, some of these philosophers argue for the broader, practical applications of their work (Ruben, 1993). This section is an attempt to do that by drawing on these analyses of best practice explanation.

IEFs are typically designed based on the *ad hoc* intuitions of system developers or on empirical studies of how humans interact in an explanatory dialogue (Cawsey, 1992). Clancey (1983) argues that explanation frameworks, such as the "epistemology" developed as part of the MYCIN project, are useful in helping to direct the knowledge

acquisition, knowledge representation, and explanation delivery efforts of systems development groups. New perspectives on the engineering of knowledge help to progress the field by stimulating the development of new hypotheses and by explicating the potential of new areas for research, (Gregor & Benbasat, 1999; Stefik & Conway, 1982). A central feature of an explanation is information *about* the information being provided in response to the explanation request (Draper, 1988). Why and how a particular piece of information relates to the explanation request, and why a particular element of information should be considered reliable are both important elements of the meta-explanatory information that help to frame and ground an explanation statement.

The challenge to designers of systems capable of computer-generated explanation is to develop a framework for explanation that is both meaningful to human users and amenable to computer manipulation. As early as 1945 (Bush, 1945), scientists were bemoaning the lack of fit between natural languages and mechanized languages. Even then it was recognised that by modifying natural language to be more amenable to device manipulation, the meaning of the language at once becomes less intelligible to human users. According to Wenger (1987, p. 277):

> *"Knowledge communication requires a form of epistemological mapping whereby the system's representation of knowledge and processes reflects the human approach to the domain."*

Smith (1998) points out that there are two ways to view this issue: as a user-friendliness problem, and as a computer-friendliness problem. User-friendliness involves making computers smart enough to interact with users on their terms. Computer-friendliness involves users showing some degree of empathy to the limited language formats available to computers. Computer-friendliness involves consistent encoding of information to make texts easier to interpret by computer, using tools such as meta-data, mark-up languages, and formalized languages. Smith somewhat polemically argues that while most efforts to deal with this problem have focused on making computers and their languages more usable, as complex devices computers require some basic competencies from their users in order to achieve their potential. Efforts to develop IEFs and improve the explanations they provide occupy the problem space as described by Smith. The approach developed in this dissertation is

an attempt to both improve the computer-tractability of explanatory knowledge and provide computer users with an additional tool to develop their own competencies.

3.3 Aspects of Explanation

Among the attributes that are most often related to the concept of explanation, good explanation or complete explanation in the philosophy of science are explanatory power, predictive content, simplicity, and theoretical unification (Gasper, 1991). Friedman (1988) enumerates three essential requirements of a formal theory of explanation:

1. It should be general.
2. It should be objective; it should not rely on the strength of particular science that is in fashion.
3. It should relate explanation to understanding.

Despite these attempts to enumerate some general principles for describing and evaluating models of explanation, these ideas are subject to intense debate among philosophers of science and all are far from achieving any degree of unproblematic acceptance. The question of what constitutes explanatory power is central to these debates and is difficult to consider meaningfully outside the context of a specific theory. Whether or not an explanation may also serve as a prediction is again dependent on the theoretical frame of reference; as will be shown, some approaches reject this criterion outright. The idea that simpler, more general explanations are preferable to more complex ones seems particularly appealing. However, as discussed further below, even this criterion is subject to doubt. The following sections briefly review some of the concepts seen to be most central to the debate over these theories.

3.3.1 Causality

Statements of causality are often considered to be among the potentially most powerful elements of the explanation equation. Some philosophers claim explication of causal relations as the central task of any theory of explanation (Salmon, 1975). Unfortunately, scientists and philosophers are unable to agree - especially among themselves - on what causality really means (Pearl, 1996). As Pearl points out, in

ancient history causality was unproblematic, only the gods, and possibly humans and animals depending on the prevailing view towards free will, were capable of causing anything. It is only due to the success of science in identifying correlations between entities in the natural world that causality has become such an entangled topic and the profusion of complex engineered artefacts has only increased the density of the world's causal web.

Causation is sometimes defined as one event necessitating, having the capacity to bring about, or providing mechanisms for the production of the second event (Gasper, 1991). These mechanisms or processes may be categorised as mechanical, biological, cognitive, or social in nature (Chandrasekaran, et al., 1999). Churchland (1989) argues that human understanding of causality is based on *etiological prototypes*, that is, sequences of events described in the abstract that help us to understand how events are etiologically and temporally related.

The essential critique of our ability to determine causation is still Hume's (1888, as elucidated in Salmon, 1998). To Hume, perceptions of causality are derived from three elements: the *temporal priority* of cause to the effect, the *spatiotemporal contiguity* of cause to the effect, and the fact that whenever the cause occurs, the event follows (*constant conjunction*). However, Hume argued that any attribution of causal force is based solely on observed correlations rather than observed phenomena and is therefore significantly flawed. Humean counter-examples include scenarios such as a barometer falling and the coming of a storm, or a cock crowing and the sun duly rising. In both of these examples the events fit the pattern seen to describe causal relations, yet clearly a falling barometer does not cause the storm any more than the cock crowing ensures the sun will rise (Pearl, 1996). The crucial problem is that hypothesised causal dynamics that may hold between events are often unobservable and therefore unacceptable in an empirical view of science (Gasper, 1991).

In a more relaxed and pragmatic view of causation with specific reference to the social sciences, Huberman and Miles (1994) argue for the central role of temporality in explicating causal relations. In their view, narrative causal accounts can be used to identify the loop-back mechanisms by which event A causes B, which in turn affects the nature of event A1 which causes B1, and so on. They identify, following Hill's

(1965) discussion of causation in epidemiology, a set of factors that should be included in the causal equation.

1. Strength of association – much more B with A than with other possible causes
2. Biological gradient – if more A, then more B
3. Coherence – the A-B relationships fits with other knowledge about A and B.
4. Analogy – A and B fit the well established pattern noted in C-D

Miles and Huberman argue that causal relations must be considered as part of a potentially immense network of factors where all nodes have the potential to affect all others. In the complex socio-technical milieu of information systems development and use, this network is likely to be large indeed. The rest of this chapter will discuss the role of causality, if any, within each of the theoretical paradigms presented. Section 3.2 will begin a description and demarcation of causality as it relates to explanation of information systems that is further developed in later chapters.

3.3.2 Generalisation & Unification

A conception of explanation derived from the received view, the deductive-nomological (D-N) model of explanation (described below), is that of explanation as simply the reduction of the unfamiliar to the familiar through the subsumption of lower level, detailed theories into more general ones (Friedman, 1988). According to Friedman, this model of explanation is one which claims that at any point in time, science holds a set of base concepts which are 'pure' in their predictive power, they are well tested, base units of knowledge which themselves require no explanation and which can be used to explain other phenomena. Ruben (1990) argues that this thread of "ultimate explanations" has been considered by, among others, Plato, Aristotle, and J. S. Mill. In their view of "ultimate explanation", Plato and Aristotle hold that there is such a thing as a non-demonstrable understanding, which is an ultimate explanation. Mill argues that these ultimate explanations are themselves inexplicable - we could never really know what was an ultimate explanation, and what would someday be explained and subsumed under another general law.

Scriven (1988) postulates that the act of explanation involves relating phenomena outside a given set of understood phenomena to those inside the set. Presumably, those inside the set are simpler and more general. Though this theory is appealing as an attempt to relate explanation to understanding, its reliance on the idea of an explanans (that which does the explaining) as essentially simpler than the explanandum (that which is explained) fails when applied to complex fields in science. Friedman (1988) argues instead that explanation is not required to be familiar, it merely needs to explain, and cites examples from the laws of physics in which the phenomenon being explained is far more familiar then the explanation itself.

Ruben (1990), drawing from Plato, Mill, and Aristotle, argues that the alternative to ultimate explanations is a "vicious" form of infinite regression. Because we could never have a truth on which to base our beliefs, none of them would be ever be truly justified. In a Kuhnian counterargument, Friedman (1988) points out that any set of ultimate explanations relies on the "intellectual fashion", and is therefore sceptical of their role as the units of knowledge upon which all explanations are based. While our ability to arrive at any philosophically valid instances of these ultimate truths may be in doubt, it seems that some sort of 'stopping mechanism' is at work in the process of everyday explanation. Various theories deal with this mechanism differently, and some simply ignore it altogether. The question of how deep might be deep enough in the context of IS development and use is addressed later in this chapter.

3.3.3 Explanation, Prediction, and Description

One of the more widely discussed debates in the field of explanation theory is that of the *symmetry thesis*. This thesis holds that there is only an epistemic, pragmatic, or temporal difference between explanation and prediction, there is no logical difference between the two concepts (Ruben, 1990). Salmon (1993) describes explanations of the deductive-nomological kind (outlined in the next section) as providing *nomic expectability*. Scriven (1988) argues against the idea of explanation as "essentially similar" to prediction. Although understanding something may allow us to predict it, being able to predict it does not constitute understanding. Central to his argument is that a prediction must make a statement on when something will happen or what will happen. A causal explanation only has to describe what made an event happen.

Scriven argues that in explanation we know an event has occurred and this adds greatly to the understanding of the event. In prediction we do not know the event has occurred and so lack this critical evidence towards understanding. Hempel (1942) argues that this distinction between prediction and explanation is pragmatic; a prediction may be provided for an event that has not yet occurred and explanation for an event that has occurred. He acknowledges that in practice, however, because explanations are often given in an incomplete form, they do not always provide enough information to be predictive. The notion that what constitutes a good explanation should also serve as a prediction is fundamental to the classical methods of the natural sciences.

In the social sciences, explanations may be given to prescribe future behaviour, or to describe and justify behaviours in the past. In the latter case these often take the form of *inference to the best explanation*, also known as abductive explanation (Harman, 1986). Abduction is used to select from among competing explanations of a singular event, and is often used to explain how scientific theories are applied and how people make attributions regarding the behaviour of others (Thagard, 1999). An interpretive or hermeneutic view of explanation is abductive in the sense that what suffices as an adequate explanation is the account that makes the most sense of a given situation (Ruben, 1998).

A second form of the symmetry thesis attempts to equate explanation with description. This thesis holds that to explain an event or a law is to describe it, and vice versa. Scriven (1988) discourages the idea of explanations as descriptions, though he concedes that the right descriptions, those that fill a gap in a person's understanding, are often good explanations. Ruben (1990, following Achinstein, 1983) argues that the same explanatory content can be conveyed as an explanation or as something else, for example, a criticism. These debates can seem counterproductive in that they attempt to explain one ambiguous concept, explanation, with another, description. For our purposes, we assume that explanation is a special, more rigorous kind of description but that more standard descriptions, for example terminology, play an important role in the explanation equation.

These arguments point to the essential role of delivery in explanation. Can an explanation exist without a motivation, a mode of communication, and a receiver's

interest? According to the pragmatic theory of explanation (discussed below), they cannot. This thesis is primarily concerned with the nature of explanation content and the ways in which this content can be structured into a sufficient explanation product, however, the effectiveness of delivery mechanisms must also be considered since they are crucial to the context in which the explanation is perceived.

Because in this thesis we are interested in explanations of events, in particular, designs, that have already occurred, the role of explanation in prediction will not be considered further.

3.4 Theories of Explanation

The historical and contemporary literature is rich with the work of philosophers who have struggled with providing a complete and precise definition of explanation. Most disciplines include within their corpus attempts to provide a framework for what constitutes an adequate explanation within their field (Hempel, 1942; Davidson, 1974). As discussed in the last chapter, research in the field of computer-generated explanation, and information systems in general, has not, for the most part, referenced this work. Although an explication of the full range of philosophical theories of explanation is beyond the scope of this dissertation, this section will examine a cross-section of the most influential formal theories to help inform the development of a more well-grounded conceptual framework for the creation of systems capable of generating explanations.

3.4.1 Deductive-Nomological Explanation

Arguably the most influential modern theory of explanation is Hempel's Deductive-Nomological (D-N) or Covering Law Model (Hempel & Oppenheim, 1948; Hempel, 1965). Almost all contemporary theories of explanation stand in relation to this work (Ruben, 1993; Salmon, 1993). The D-N Model describes explanation as the identification of antecedent conditions combined deductively with the application of general laws.

Within this model, the explanation equation is defined as:

L_i (general laws)

C_i (antecedent conditions or facts)

_____ (deductively entails)

P (the phenomenon to be explained)

The D-N Model describes causal explanation, the general laws and antecedent conditions that make up the explanans, that which does the explaining, these are considered to cause the explanandum, that which is explained. In the D-N Model, an explanation must fulfil both logical and empirical conditions of adequacy.

Logical conditions:

R1 The explanandum must be a logical consequence of the explanans.

R2 The explanans must contain general laws that are required for the explanandum.

R3 The explanans must be subject to test by experiment or observation.

Empirical conditions:

R4 The sentences making up the explanans must be true, that is, highly confirmed by the available evidence.

Hempel and Oppenheim (1948) provide additional metrics for the effectiveness of explanations that fit their model through the notion of systematic explanation. They describe systematic power as contributing to an explanation or prediction of a theory T as the ratio of the amount of information derivable by means of T to the amount of initial information required for that derivation. This idea is manifest in applications of the Brunswik Lens Model, with reliance on explanatory cues, to explanations of expert systems (Dhaliwal & Benbasat, 1996). The use of the term systematize refers to the process of constructing explanatory and predictive relationships between data. The structure of the covering law model not only defines explanation, but also suggests why we should value theories that fit the model, that is, because of their predictive ability (Gasper, 1991).

One of the major problems with the D-N Model is the determination of what counts as a law. Hempel supplies the following definition.

> *"By a general law, we shall here understand a statement of universal conditional form which is capable of being confirmed or disconfirmed by suitable empirical findings."* (Hempel, 1942)

The reliance on the presence of a natural law in a causal relationship has been a tenet of scientific practice (Gasper, 1991). In the social sciences, the notion of laws as central to explanation presents a potential problem given the role of intentionality in psychology and human behaviour (Davidson, 1994). Hempel allows that law may be too strong a term, for example, in his discussion of historical analysis, and suggests "universal hypothesis" as an acceptable substitute for the social sciences. Other problems with the D-N model include cases where an event fits the model, but is not explained, and cases where the event is explained but does not fit the deductive structure of the model. A classic example of the former is that the length of a shadow does not explain the height of a flagpole, though the length of the shadow, the position of the sun, and the laws of geometry can be combined into an explanation to fit the D-N model.

Applying a neurocomputational approach, Churchland (1989) questions how realistic the D-N model and its derivatives are, given the way in which humans seem to operate in the world. He claims that when asked to provide explanations of their actions, people seldom produce explanations that conform to the model, in particular, they seldom reference the role of laws as justifications for their behaviour. Churchland argues that the time needed to produce a proper D-N explanation suggests this is not the approach employed by people who seem to make sense of thousands of stimuli in any given waking hour. What is unclear is whether sense-making of the sort Churchland describes corresponds to the depth of understanding that is the goal of more comprehensive explanations.

Probabilistic Versions of the D-N Model

A closely related model of explanation is the class of probabilistic explanations, which attempt to address scenarios where causal events stand in a probabilistic relation to their effect or effects. Examples of probabilistic explanations are often given in medicine, where exposure to an infectious agent is given as the cause and

explanation of someone being diseased, but not all those exposed to the infectious agent actually get the disease (Salmon, 1993). Two major versions of probabilistic explanation are Hempel's modification of the D-N Model to account for probabilistic relations, the Inductive-Statistical or I-S model (Hempel, 1965) and Salmon's Statistical-Relevance, or S-R model (Salmon, 1993).

In contrast to the D-N model, which shows that a given event occurred with absolute certainty, the I-S model shows only that given the laws and antecedent conditions, it was highly probable that the event was to occur (Little, 1991). A key component of the I-S model of explanation is the requirement for maximal specificity. This requirement holds that an explanation of the I-S form is invalid unless all statistically relevant facts are included in the set of antecedent conditions. For example, an I-S explanation stating that an aircraft of a certain type with a given amount of fuel had a 0.9 probability of travelling over 500 miles would be false if the additional fact that it was flying into a 30 knot headwind was omitted.

A problem that arises in this account and which is addressed by Salmon (Salmon, 1993) in his Statistical-Relevance model, described below, is that the requirement for maximal specificity does not take into consideration the relevance of all of the available facts to the explanation that is provided. The Statistical-Relevance (S-R) model of explanation exists on two levels. The first level, statistical relevance, describes the network or matrix of factors and their associated probabilities that make up the explanans. The second level describes the causal force of this array of factors. Salmon (1984) believes that the role of general laws and deduction in explanation should be secondary to the explication of causal relations. The S-R model also differs from the I-S model in that it does not include Hempel's requirement that the explanandum be highly probable given the explanans. The model instead states that a given factor is relevant to the explanans if its presence increases or decreases the likelihood of the explanandum event occurring. For example, the probability of the average person (A) developing skin cancer (B) is not equal to the probability that a person who spends five hours each day sunbathing (C) will develop skin cancer.

The central concept in the D-N model of explanation is the role of laws and law-like statements in explanation. Strict views of scientific explanation claim that an explanation has occurred when an event is shown to be an instance of a phenomenon

governed by a law or law-like regularity (Strasser, 1985). Conceptions of laws as "ultimate truths" are philosophically problematic, and their use in the everyday explanations that humans construct is questionable. Nonetheless, laws play an important role in justifications when given as reasons in answer to a *why* explanation request. Later in this chapter we will explore the kind of laws that might be useful in explanations of IS.

3.4.2 The Pragmatic Theory of Explanation

The pragmatic theory of explanation frames explanation as "an interest-relative notion" (Putnam, 1978). The most influential proponent of pragmatic accounts of explanation is Bas van Fraassen (1988, 1991). In van Fraassen's view, explanation is best conceived as answering a given why-question from a particular aspect within a particular context. He claims that in answer to the question "why is the light on", both "because I flipped the switch" and "because we are expecting company" are explanations, depending on the motivation for and context of the original question. To van Fraassen (1991):

> *"Which factors are explanatory is not decided by features of the scientific theory but by concerns brought from outside."*

Van Fraassen employs an example commonly used as an argument against the D-N theory. In this example a valid D-N explanation is constructed to explain the height of an object (usually a flagpole), using the length of its shadow, axioms from geometry, the laws of planetary motion, other relevant facts and laws, and deductive inference. Clearly using the length of its shadow as the central fact in an explanation of an object's height is questionable. However, a counterexample drawing on van Fraassen's theory would claim that the length of the shadow does explain the height of the flagpole, if the pole was designed to cast a shadow over a veteran's grave on the day and hour of his death. These ideas are central to van Fraassen's view that theories of explanation cannot be analysed or evaluated based on innate notions such as simplicity, predictive strength, truth, or empirical adequacy that deny the role of context.

Other concepts central to van Fraassen's theory include the role of *relevance relations* and *contrast classes* in explanation constructions. Unlike the similarly named concept employed in Salmon's S-R model, van Fraassen's relevance relations describe events

that relate to the event to be explained in terms of the relevance of those events to the *purposes* of the explanation rather than to the probability that the explanandum event actually occurred. Draper (1988) argues that a sort of contract exists between those engaged in an explanatory dialogue, and that an element of this contract is that the explainer will endeavour to limit the information provided to that which is relevant to the explanation request.

Contrast classes provide information on why a particular event occurred instead of, or in relation to, another in its contrast class. Van Fraassen argues that every why question may be translated as why x instead of y? In a set of probabilistic explanations, the explanation with the highest probability is considered more explanatory than those with lower probabilities in the selected explanation's contrast class. Lipton (1990) argues that contrast classes (explanatory "foils") play an essential role in pragmatic, causal explanations by focusing an explanation on the most salient causal factors relative to the explanation request. For example, if in answer to the question "Why did you study at the LSE?" I begin an answer with "Because at American universities..." versus "Because at Cambridge...", some of the most relevant causal factors are brought into focus. The concept of contrast classes plays a central role in the explanation framework developed in this thesis. Much of the next chapter will address the role of contrast classes in the domain of IS explanation.

3.4.3 Functional Explanation

Functional explanations attempt to provide arguments for the existence or persistence of entities (objects, events, or institutions) by reference to the effects, generally the beneficial effects, of those entities. Arguments for the validity of functional explanation are controversial (Kincaid, 1990) and are especially prevalent in the fields of biology and in the social sciences (Ruben, 1993). The attraction of functional explanation, especially in social science, is based on the assumption that phenomena must have some meaning, that things exist and events occur because of their beneficial consequences, because of some *reasons* (Cummins, 1975; Elster, 1985). Functional explanations are especially relevant to engineered artefacts, where it is assumed that human design activities and workmanship are applied to serve some purpose.

Dore (1961) has examined the concepts of function and cause in an attempt to identify ways in which an analysis of functions can be translated into a statement of causality. In his analysis he differentiates between the cause of something (in his example, social institutions) occurring and something persisting. There is a distinction between the causes of the events that lead to something being created, and the causes of the events that lead to something persisting. This analysis echoes ideas from socio-biology, such as Dawkins' *memes* (Dawkins, 1989) and Sperber's *cultural cognitive causal chains* (Sperber, 1996), which both relate to the transmission and persistence of ideas within and across cultures. Dore argues that it is possible for causal events to be identified as functional. His example involves a discussion of the development of Chinese communes, stating that it is probably possible to find the minutes of the meetings and other documentation of events that led to their creation where the creation was in constant reference to the intended consequences of the communes, their function.

Elster (1985) argues that fields in social science have sought to fit the model of natural selection and environmental adaptation to the phenomena under study in their respective fields. He outlines a semi-formal model of functional explanation as an attempt to explicate the causal dimension of the theory:

1. Y is an effect of X
2. Y is beneficial for Z
3. Y is unintended by the actors producing X
4. Y – or the causal relation between X and Y – is unrecognised by the actors in Z
5. Y maintains X by a causal feedback loop passing through Z

Elster cites the Chicago School of Economists as an example of this sort of explanation successfully applied. Members of the School argued that the profit-maximizing behaviour of firms could be explained using a natural selection model in which firms survive because the rules of thumb they employ in business decisions just happen to be closest to profit maximizing behaviour. However, there are issues with part 4 in the model above that are only explained if the behaviour of the firms is

spread through takeover, not imitation, as this would entail recognition of the optimising behaviour by the imitating firms. Organisational ecology is a recently developed form of analysis that applies this evolutionary metaphor to the survival rates of organisations (Kincaid, 1990; Hannan & Freeman, 1989). Parts 3 and 4 of Elster's model are problematic when applied to situations where intentionality plays a role in the production of Y, for example in the design process. Elster calls explanations that drop part 4 from the model *filter* explanations; the beneficiary of the behaviour is able to reinforce and cause the behaviour to persist even though they never intended for it to emerge.

Several arguments against the idea of functional explanation have been identified and include problems such as a lack of supporting evidence for the mechanisms by which certain features or practices exist (Kincaid, 1990). Since these are often not identified, functional explanations are not generally falsifiable. Another problem is that functional explanations do not show how a given feature came to exist, only how it persists. Against the former, Little (1991) argues that in a complete functional explanation, the causal feedback process, the missing mechanism highlighted by Kincaid, must be identified in order for the explanation to be fully coherent. Against the latter, Little urges us to consider a given feature in terms of its current effects, not any future effects that the feature might have. A given feature's current beneficial effects are what cause it to persist and to disperse and any beneficial future effects are an outcome of this persistence and dispersal.

3.4.4 Rational Choice and Explanations of Action

Rational choice theories of explanation attempt to describe behaviour, especially human behaviour, in terms of the perceived benefit of that behaviour relative to other possible behaviours. As pointed out by Simon (1996), such theories of behaviour are often simplistic given the ambiguities of the beliefs and desires that purportedly drive rational choice decisions. In his earlier work, Davidson (1963) argued that rationalisations, an actor giving reasons to explain why a particular action was taken, are a type of causal explanation. In particular he claims, "the primary reason for an action is its cause." Central to Davidson's argument was the idea that primary reasons express the intention of the person performing the action. When we give a reason for taking an action, and the reason is true, we expose the belief or attitude that is causing

the action. Davidson counters arguments that reasons are not causes of actions by appeal to the events that led to the formation of the beliefs that underlay reasons. An event-belief-reason sequence constitutes a causal chain, events lead to belief formation, these beliefs, now given as reasons, cause an actor to take particular actions.

In later work however, Davidson (1974) seems to agree with Simon that it is impossible to identify the true causes and effects of human behaviour or actions. Any formula that attempts to identify how one consistent set of human beliefs and desires emerges as the causal factor from a possibly vast array of competing beliefs and desires will simply not be useful for predicting or explaining actions. The role of intentionality in any such decision analysis will always serve to confound such analyses. Competing beliefs that impact choice of action in a given scenario are selected based on a weighting that is not necessarily predictable or even reliably describable. In answer to the received view of D-N explanations, Davidson allows that laws may be involved in the development of event-belief-reason chains, but this involvement may be so far removed from belief formation as to be irrelevant to the explanation.

Attempts to fit human action to the D-N model of explanation, to claim that these actions are governed by laws, have not met with great success (Ruben, 1998). In the case where a form of rational choice theory is used as the requisite law, these explanations fail to account for many actions, for example, those that are commonly attributed to "weakness of will" such as smoking. Though additional antecedent conditions can be added to make the explanation work (given rational choice theory, given the dangers of smoking, and *given that the person is rational*, the person will not smoke), Scriven (1956) argues, following Hayek, that D-N explanations in social science are made potentially intractable by the large number of potentially "useful concepts" and the fact that many *implicit* laws are used in these explanations.

3.5 Theories of Explanation for IS

This section derives from the preceding discussion a framework that describes how the essential elements of philosophical theories of explanation may be applied in the IS domain. Recall from Chapter 1 that the intent behind this effort is one of design

science and the goal is to establish a more theoretically grounded, less *ad hoc* conception of the information needed to explain an IS in its socio-technical context.

A common approach to explaining how a device, a system, or even an organism works is to first break it down into the particle level, the component level, and the functional level (e.g., DeMarco, 1979; Dennett, 1987). This mechanistic, reductionist approach to explanation structure involves showing how a device's components add up to produce the functional characteristics of the device. The reductionist approach is a foundation of the system sciences, and is represented in the plethora of IS development methodologies that are based on stepwise techniques (e.g., Boehm, 1988). This view is in contrast to the holistic systems model, where the system is seen as only a part of a larger, socio-technical framework with its own emergent properties being irreducible (Lin & Cornford, 2000; Mumford, 1996). Holistic accounts appear in IS explanations corresponding to the functional or pragmatic types, where the purpose of the system being explained, or in the latter case the purpose of the explanation itself, are seen to override the utility of systemic, causal interactions. This tension reflects only one of the many challenges that emerge when attempting to resolve the computer versus human language problem.

3.5.1 Causal Relations in IS

One of the most powerful attributes of the digital computer - and one of its most significant problems - is the degree of freedom available to the system designer. Computer programs, unlike traditional design disciplines such as architecture for example, do not face the physical, material constraints of most other design disciplines. Instead of theory describing reality, computer programs in essence allow theories to drive the creation of reality (Kay, 1977 cited in Rheingold, 2000). The linkages between action and effect become increasingly complex and obscure to users as the amount of engineering involved increases (Barber, 1988). An example of what Barber describes as an obscure linkage, one that results in an incompatibility between the user's mental model and the actual device model, is that of using the tiller to steer a boat. As the tiller is pushed one way, the boat turns in the opposite direction. Some accounts suggest that the human (and some animal) ability to construct *causal maps* is part of our innate cognitive architecture (Gopnik et al., 1999). Barber argues that such causal maps, or mental models, are developed by humans to visualize the direct or

indirect linkages between an action and its effect. Without the visible, tactile feedback common to most controls that humans encounter, the tiller for example, these mental models are more difficult to develop and operationalise.

A benefit of computers with modern interface technology is that though almost *all* such linkages are hidden from the user, any action-effect linkage can be designed into the information displayed on the screen or otherwise fed back. The paradox is that these feedback mechanisms must be explicitly designed and programmed; a directed effort must be made to expose every cause-effect linkage that is initially hidden. In developing these feedback mechanisms, the system developer introduces yet another device into the system, one that might result in additional mental model incompatibility. The design and development efforts meant to expose these hidden linkages are among the most important to explicate in an explanation facility meant to show the cause and effect relationships at work in a system.

What is called deep knowledge of a complex domain or artefact is often considered to be the causal knowledge that underlies it. Clancey (1983) identified this causal information, in the form of a process model, which he refers to as a causal chain, underlying the program execution, as an essential component of the support knowledge that underlies a KBS design. Lamberti and Wallace (1990) argue that in the context of ES explanations of problem solving strategy, providing system users with information about the causal relations between system elements is a key tool in the development of users' knowledge in the problem domain.

In an information system, this deep knowledge includes the logic and control data (in an expert system, the production rules). Causal knowledge in the form of A causes B is really no different from IF B THEN A. (Chandrasekaran, & Mittal, 1983, p. 433). However, to explain a system requires relating these internal causal forces to the domain in which the system is used. One element of this relationship can be explicated by reference to the *design rationale* that relates a system's logic and control data to the problem domain. In addition, Chandrasekaran, et al., (1999) highlight the importance of side-effects when considering causality in the context of engineered systems. However many scenarios might be considered during the design of a system, we can never be sure that the next one to emerge (perhaps from the use

context, after design) will not invalidate the current explanation of the causal relation that we think exists in a given domain.

Previous expert system IEF research has acknowledged the role of *explicit* representations of the causal relations between elements of a problem domain in providing justifications for program behaviour (Swartout, 1983). As in nature, the unobservable status of many causal forces at work in an information system, both those of its internal functioning and those that led to the design of those functions, makes their explication difficult. But, as pointed out by Robb & Brown (1987) it may be that those factors not easily observed are of the greatest explanatory value, not least because they motivate most explanation requests.

Brown (1984) claims that an expert's ability to handle the unexpected is one of the things that makes them expert. Experts draw on deep, inferred causal models instead of experiential knowledge when dealing with new and novel problems. For the developing expert, Nathan (1992) uses prior empirical research to argue that students learn most effectively when they are forced to reason about the underlying causal relations in a domain. As discussed in the last chapter, researchers and practitioners applying the minimalist approach to system documentation design also argue for the efficacy of this kind of exploratory learning. Later chapters will develop an approach to providing tools in support of this learning mechanism.

3.5.2 Unification & Generalisation

Draper (1998) argues that a key to helping a user understand how a system relates to its domain is the way in which the top level, most general structures (such as menus) are identified and related to the function of the tool. Generalised explanations such as these are often the most widely used, and generalised headings in a documentation set are regarded as the most effective indexes to more detailed information (Van der Meij & Carroll, 1998). The use of generalised headings combined with the ability to obtain successive levels of detail is a standard, proven architecture for delivering usable online help in a software application (Patrick & McGurgan, 1993).

In the field of AI, many expert system researchers have argued that for a system to provide sufficient explanations, it must include knowledge of first principles in the domain (Hayes-Roth, et al., 1983). Concepts considered first principles for a given

domain are seen to constitute the "ultimate explanations" or "ultimate truths". However, strongly reductionist approaches such as this provide no guidelines for identifying these basic knowledge elements. In the D-N model, the laws in a valid explanation take on the localised status of a first principle, but are themselves generally seen to be subsumed by a still more basic law. The pragmatic account would deny the role of first principles in an explanation unless their provision is seen as central to the explanation request.

Stefik and Conway (1982) present an approach to knowledge engineering that employs a set of three criteria for driving the structure of a knowledge base. In what they argue is an attempt to correspond to our predilection for simplicity in scientific endeavours, these criteria are as follows:

1. *Basis* simplicity – the number of basic elements in the ontology.
2. *Expression* simplicity – the length of the average or most common expressions.
3. *Composition* simplicity – the number and simplicity of the rules used to combine terms.

Though these criteria represent useful heuristics in the design of the knowledge base, Stefik and Conway point out that a tension exists between the desire for simplicity and the need for coverage, or relative completeness, in the knowledge base content.

Information systems analysis and design formalisms rely to a very great extent on the process of classifying and generalising about both process and data types. The process of creating generalised abstractions that encapsulate both the information and behaviours of a given domain entity is central to the systems modelling approach. The creation of generalisation/specialisation hierarchies is one of the most important techniques in modern, object-oriented design (e.g., Booch, 1994; Fowler & Scott, 2000) and clarity, simplicity, and unity are considered to be among the most important attributes of computer programming languages (Pratt & Zelkowitz, 2000). However, these analytical processes necessarily result in increased uncertainty regarding the true relationship of the system design to domain principles (Parnas & Clements, 1986). These arguments highlight the role played by generalisation and unification in the design of IS, and suggest their explication as a significant role of an

IS explanation facility. The role of generalisations that make use of laws or law-like statements is discussed in the next section.

3.5.3 D-N Explanations of IS

The major contribution of Hempel's Deductive-Nomological model is the role of laws and law-like statements in the explanation equation. Assuming Hempel's less rigorous definition of such laws, we may admit a range of facts that can contribute to the explanation of information system operations including: the physical constraints of computing and networking hardware (physical laws), constraints introduced by the syntax and semantics of a particular specification or programming language (logical laws), published technology standards, development and user-interface guidelines well-grounded in empirical studies, laws in the form of regulatory frameworks that may bound the development and operation of systems in safety critical domains, and business rules, for example, accounting standards and intra/inter-organisational business process specifications that affect the finished IS product. Professional and organisational norms, such as those published by the IEEE and the ACM, which prescribe how professionals behave in practice may also be taken as a form of law or law-like statement. In the socio-cultural context, conventions of behaviour such as those applied in a business meeting versus those applied in a pub grow out of the simple human need to coordinate their actions (Ruben, 1998). In the context of information systems development and use, these norms may act to guide individuals in their work. Considered in this way, laws take on an importance in the explanation structure in that they provide a reference structure that helps to answer questions about design decisions that impacted the system model.

Technological artefacts such as information systems and their components have been referred to as objects with a dual nature: one physical and the other functional (Kroes, 1998). The physical dimension of information systems may be seen primarily in the constraints that act upon such a system, for example, hardware, software, and telecommunications capabilities. A second element of this physical dimension relates to the ergonomics of a system and its fit with human psychological and physical capabilities. Unlike many other engineering disciplines where material constraints play a major role in the form of the finished artefact, information systems, especially software, are bound by few such physical constraints. Logical, intellectual, and social

interaction (e.g., team dynamics) constraints play a far more prominent role in the shape of the finished IS product.

Many social 'facts' achieve the status of a law in society simply because a society or culture has agreed on a shared or standard meaning, for example, consider the concept of money (Ruben, 1998; Searle, 1995). IS development and operational (use) standards, methodologies, and norms emerge from a similar process, but are more contingent than many other social laws because the environment in which they apply is evolving so rapidly. Nonetheless, for some period of time the laws may act to constrain how a system is developed and used and therefore help to explain their form and behaviour.

What qualifies as a law in the domain of the social sciences frequently takes the form of a prescriptive statement or one that defines an ideal type as a goal state. Many information systems development practices are driven by the application of standards produced either internally to guide the development project process and deliverables, or externally in cases where regulatory documents play a role in the development process (e.g., safety-critical systems and systems developed to government contract specifications). Consider an IS product developed for the European Community (EC) market and subject to EC standards related to ergonomics. This sentence consists of the antecedent conditions, that the software is being developed for the EC market, and when combined deductively with the law-like statement that is the EC published ergonomic standard, we have an explanation of why certain decisions were made regarding the functionality of the resulting system.

System experts who do not possess the ability to draw on a base of increasingly detailed and experiential information in the course of providing an explanation are frequently the subject of credibility problems, especially in safety-critical domains or those that are considered to have a rich, intellectual content (Chandrasekaran & Mittal, 1983). In order to successfully explain a concept at some level of expertise, a clear model of the domain knowledge and a rigorous understanding of the problem solving process is required (Hasling, et al., 1984). By drawing on established laws or their empirically well-grounded equivalents in a given field, experts establish this credibility in the explanations that they provide.

Hempel (1942) argues that explanations in the social sciences (specifically, history and sociology) often exclude statements of general laws. They are considered implicit and relate to individual or social psychology that, it is assumed, are familiar to the person to whom the explanation is being given. Though laws may be excluded from the explanation structure, they can be called upon to add credibility or justifying grounds to the assertions of the explanation itself (Salmon, 1993). This conception of explanation as including assumptions about the state of the person receiving the explanation is central to the next theory to be discussed.

3.5.4 Pragmatic Explanations of IS

As discussed in the last chapter, a key component of human-to-human 'everyday' explanation is that people are able to quickly establish the motivation for an explanation request and at least a portion of the background knowledge possessed by the requestor and apply this information to the construction of an explanation (Winograd & Flores, 1986). Earlier in this chapter we discussed how this informs the pragmatic theory of explanation, which prescribes knowledge of interest-relevance and the use of contrast classes to help vector explanatory content for the situation at hand.

Despite the intuitive attraction of providing for the pragmatics of explanation delivery, the problems associated with user modelling and a system's ability to perceive context present significant obstacles to achieving this goal. In the user modelling area, the level of complexity involved in classifying users according to some system-use related criteria presents a daunting challenge (Allen, 1990) and some researchers even question the desirability and ethics of capturing the amount of personal information required for the task (Browne, 1990). The problems inherent in providing a computer system with the ability to recognise context has been highlighted in the literature, perhaps most effectively by Dennett's paper (Dennett, 1990), which describes the now famous Frame Problem discussed in the last chapter. This problem can be somewhat mitigated by providing explanations in a format in which the explanation user has control of the information that they extract based on their own knowledge of context rather than the system's. One way of presenting information in this way is through hypertext trees, where information is organised into categorised 'trees' that present information in successive levels of detail and allow

branching based on immediate information needs. The information selected for display may be culled based on the user's context as determined by their position within a program, and either simplistic user stereotypes or user information display preferences. Information selected and structured in this way allows users of information systems to make their own selections based on *their* perceptions of relevance to current information needs.

Relevance Relations

Though pragmatic theories of explanation highlight the importance of audience and context in any account of explanation, the problems inherent in attempting to approximate human expertise in explanation provision suggest that rather than attempting to mimic human behaviour, researchers in computer-generated explanation should consider redirecting their efforts such that the strengths of computers and software systems are best leveraged. As Gregor and Benbasat (1999) point out, established theories related to cognitive effort suggest that users of an information system will not use explanation facilities unless they perceive the cost of retrieving and processing these explanations to be outweighed by the benefits accrued from obtaining them. Here, the importance of the relevance relations as described by van Fraassen and Salmon becomes critical. Established models of computer system documentation suggest that users of information systems are parsimonious in their use of computer system documentation (Carroll, 1990), and that expert users of complex systems will avoid using documentation seen as superfluous to their immediate needs (Mirel, 1998; Hackos, 1998). They seek out information that will assist them in achieving their immediate aims, and quickly give up their search if they do not succeed quickly, preferring instead to explore the systems functionality until they are able to achieve the desired end.

Contract Classes

The idea that contrast classes, or what could have been rather than what is, is one of the central points in van Fraassen's account of explanation (van Fraassen, 1988). Contrast classes are used in explanation to vector a concept in relation to its possible alternatives. Providing an account of why a system feature or process was implemented in a certain way is sometimes best expressed in terms of the alternatives and the reasons why they were not selected for implementation. At least one expert

system IEF includes access to "why not" justifications as part of its rule base (Whitley, 1990). The role of contrast classes in IS explanation is one of the focal points of the next chapter.

3.5.5 Functional Explanations of IS

Pearl (1996) argues that with the development of increasingly sophisticated engineering techniques, the products of these techniques became "carriers of force, will, and even purpose." Functional explanations are perhaps most obvious in their relation to the development of technological artefacts such as information systems (Little, 1991). Though problematic in the natural sciences, functional explanations must play some role in a discussion of artifacts, which are *designed* to serve some *purpose*. They are one of Kroes' (1998) two aspects of technological explanations: the physical and the functional. Information systems are (usually) developed to solve a particular problem. Their structure is related to their end goals and the features and processes the system must implement in order to achieve these goals. Kroes describes the design process as that which translates the commercial requirements for an artefact into a description or blueprint for its physical structure. During this process the functional aspects of the artefact are translated into a set of physical characteristics that will achieve the desired functional, and presumably commercial, goals. While Kroes claims that there is little relation between the physical description and the functional description of the artefact at the end of the design process, he argues that a complete design must include a technological explanation which he describes as "an explanation of the function of a technological object in terms of the physical structure of that object" (Kroes, 1998).

Kroes excludes from his analysis an important element of a software system design, namely, a logical description of the artefact as distinct from its physical structure and its functional description. In a software system design, physical structure and physical constraints are largely transparent to the designer (except in cases such as operating systems, network control software and other device-dependent systems). Though the designer is likely to employ some physically determined heuristics during the design process, for example, minimise disk storage, CPU processing, and network bandwidth usage, etc., the constraints with the greatest impact on the software design process are logical. These logical constraints are in turn determined by a myriad of factors. The

limits of the mind and imagination of the software designer are among these constraints. These constraints are compounded in team design processes where the need to communicate the substance of a logical design becomes yet another constraint (Brooks, 1975). Another important constraint is the ability of the end user to understand and effectively use the artefact that results from the design. Organisational and business factors, in particular cost, also introduce constraints that become important to a comprehensive technological explanation of a designed system.

A logical description, most typically a design specification, mediates between the functional specification or requirements of a system and its instantiation. These specifications include information about *how* the system is designed to implement the functional specification, but typically not *why* this design takes a particular form. Providing justifications of a system structure relative to its domain is an essential component of explanation. The next chapter shows how these justifications can be captured in the form of design rationale and proposes a method for using these rationales in system explanations.

3.5.6 Rational Choice Theories of IS

The rational choice theory, as an archetype theory to explain human action, must be included in any discussion of technological explanation in order to account for the impact of human reason on the structure of the artefact to be explained. Despite the philosophically problematic nature of rational choice theories of explanation, individuals and organisations do, at least sometimes, make decisions based on their perceptions of the relative value of the potential outcomes. As human-designed artefacts that emerge from a complex socio-technical, organisational, and psychological context, information systems are subject to the web of decision-making that characterises the artificial. In the IS development context, a vast array of interconnected and sometimes competing beliefs and desires may interact in the course of a given design decision. Capturing the elements of such an array is likely to be practically impossible given the sheer number that may be present in a group decision context, and the fact that many decisions rely on information that is tacit, inaccessible even to the individual who possesses it (Schön, 1983). Nonetheless, decisions are made, and the primary decision factors can sometimes be made explicit within the dynamics of an information systems development project. In such cases,

rational choice explanations do fill a gap in the information systems explanation equation by allowing for the goal maximizing behaviour, however simplified by constraints of access and capture, of individuals as they work in teams within a complex web of social, organisational, psychological, and technical constraints.

Explanations of human action draw on an incredibly wide range of often competing theoretical models and empirical research programs. Explanation of action theories exist on the cusp of philosophy and psychology, and are applied in virtually all of the social and economic sciences. The rational choice theory of explanation is but one of those proposed to explain human action. Space and time preclude a full, independent explication of psychological theories of explanation here. This is unfortunate given the additional theoretical and empirical perspectives they provide on the problem. Good collections exist that deal with the question of explanation in psychology (Block, 1980) and cognition (Keil & Wilson, 2000). Because so many studies of IS development and use contexts draw on these theories, they will be least partially represented throughout the thesis.

3.6 Discussion

Based on the discussion in the previous section, a set of principles may be derived to contribute to a conceptual framework for the knowledge content of explanations and explanation-generation technologies in information systems. Deductive-nomological explanation emphasises the role of laws in explanations. Laws play a role in IS explanation by relating design decisions and the resulting system structure to the physical laws, standards, norms, and other reasonably well established universals that both constrain and guide the design process.

The pragmatic theory of explanation suggests the goal of parsimony in explanation delivery by accounting, as much as possible, for the reasons behind the explanation request and the interests of the explanation requestor. The concept of relevance relations from the pragmatic theory suggests limiting the explanation content to that with direct relevance to the system feature of interest. Among the ways this goal can be achieved is by considering the context in which an explanation is requested and by mapping explanatory content to the tasks and goals of the information system user. A second central concept from the pragmatic theory is that of the role of contrast classes

in helping to frame an explanation. In the IS context this can be achieved by relating design decisions to the options that were *not* adopted in the final design, to what could have been rather than what actually is.

Earlier in the chapter we discussed how central the theory of functional explanation is to designed artefacts. We achieve this for IS explanations by relating system design decisions to the original motivation for the system, to the tasks that the system supports directly, and to the purpose that features and processes implemented in the system are meant to serve in the system and domain context. Information systems are designed and developed to support users as they perform tasks in an application domain within an organisational and social context. Explanations of information systems should help users relate design decisions to the underlying motives behind them with reference to the system domain and context.

The rational choice theory of explanation, informed by Simon's concepts of "bounded rationality" and "satisficing" (Simon, 1996), admits the concepts of human beliefs, desires, and intentions to the explanation equation. These concepts are themselves related to other explanatory components such as the constraining laws and functional purpose that impact the design task. A design team may believe, even if the belief is false, that a particular system requirement exists, that a particular standard applies to what they are building, or that following a particular methodology will result in a higher quality system. These beliefs are combined in the design decision making process with the teams assumed desire and intent to build the system within these constraints while still meeting the functional requirements behind the effort. Capturing and communicating this process contributes to the explanation equation by exposing the ways in which the IS development team and other interested stakeholders affect the final form of the system in their deliberations.

Each of these elements contributes something unique and necessary to a complete, idealised explanation of an information system, an IS feature, or an information transformation process. Explanations conceived at this level of detail and directedness are necessary given the complexity of the information systems development and use context. They also help to guide explanation knowledge capture efforts and contribute to the tractability of the information needed by systems that attempt to include

explanation-generation capabilities. The model presented at the end of the last chapter may now be expanded to include these explanatory elements.

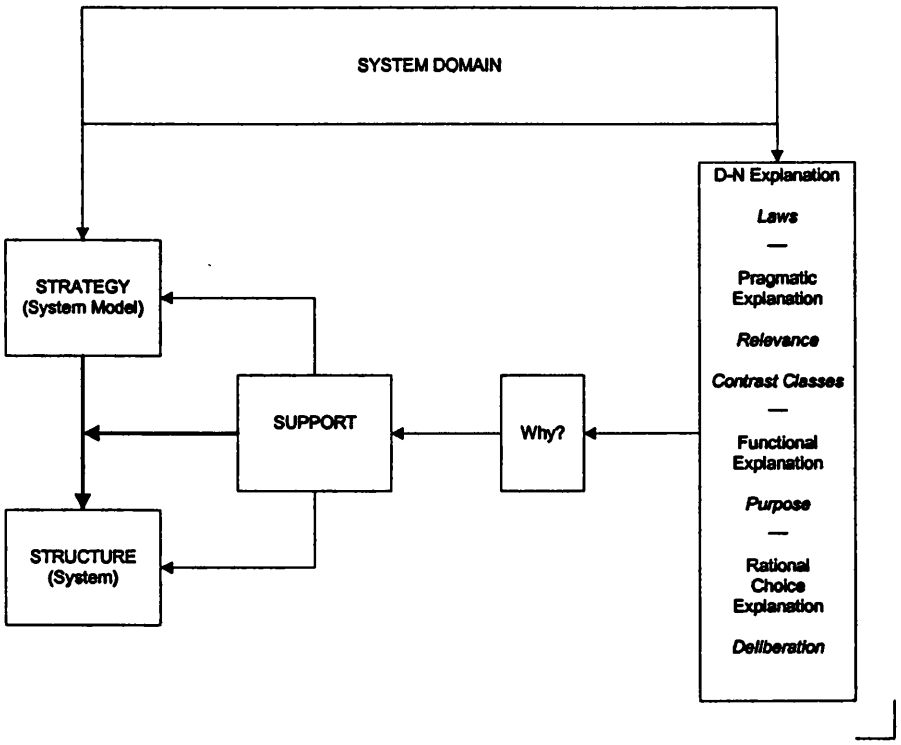


Figure 6 - A Framework for IS Explanation

In the last chapter we saw that Clancey's (1983) structure, strategy, support framework persists as the dominant model of explanation in knowledge-based systems. Structure refers to the way in which a system is constructed, the physical and logical entities that instantiate the design. Strategy refers to the higher-level problem solving approaches employed by humans in the application domain, which the structure of a system attempts to implement. Support knowledge refers to the justifications that lie behind structural and strategic entities and concepts as they exist both in the problem domain and as implemented in the system. As Clancey pointed out, the relationships between these types of explanatory knowledge are crucial to a system explanation, but are frequently made implicit during implementation.

A second, more ambiguous taxonomy employed in the KBS literature is that explanations consist of the information needed to answer *what*, *how*, *why* questions. To these we can add *who*, *when*, and *where* questions to complete the model. We need to disambiguate these terms in order to make them more useful to the discussion. *What* information is defined as a type of support information that may consist of

definitions and other terminological information. *What* information may describe entities in either the problem domain (e.g., accountancy), or in the system itself (such as menu items, button, windows, fields). *How* information is defined as "how does the system (where the system includes human, technological, and supporting artefacts such as documentation) implement a given strategy in its design?" How does the feature or group of features X support the achievement of task Y. By this definition, *how* information becomes crucial in resolving the problem of relating elements of the structure, strategy, support framework. *Why* information we consider as another class of support knowledge, one that is also crucial to explicating the links in Clancey's framework. *Why* information provides justifications for the way in which a given feature was implemented in the systems design. *When* and *where* information are forms of support knowledge that can be used to describe and justify the temporal and navigational elements of a system design. Finally, *who* information refers to the system project stakeholders in terms of how they impact system design.

From the ITS literature, especially that which is focussed on user modelling, we derive an *information limiting principle* that states that every effort should be made to ensure that the information provided in an explanation is tractable given constraints of time and cognitive resources and that this information relevant to the system user's most pressing needs. In light of the challenges faced by true user modelling, a combination of context sensitivity, where context is defined as simply where the user is in the system at a given point in time, and user stereotyping, where users classify themselves or set system preferences regarding their information needs, will be used as a substitute for user modelling to implement this limiting principle.

Approaches to system documentation include the systems approach, the task-oriented approach, and minimalism. As discussed in the last chapter, the system approach involves comprehensively identifying and defining the components of a system and as such maps most closely to *structural*, or *what* explanations. Task-oriented manuals consist of *support* or *how* knowledge designed to express the way in which domain strategies are implemented in a system, how a given set of features supports achieving some task. Minimalism as a documentation approach describes a philosophy of instructional design and learning, rather than any particular type of content. This philosophy includes the key idea that by providing *traces* of *how* and *why* support

knowledge, users can discover on their own the way in which a system was conceived to help them achieve their goals.

A good explanation of an IS is one that focuses to the greatest extent possible on the requestors information need. This focus can be achieved through context sensitivity in the IEF mechanism and by taking into account user stereotyping and user preference data when formulating the response. The information provided in the explanation should be well designed in the sense that the structure of the information provided is based on an indexing system that uses general principles to help guide users in their search. The information provided should be directly relevant to the request, but should allow for exploration of additional supporting material. In addition to providing what (terminology and definitions) and how (task related) information, explanation requestors need access to why information, the justifications that relate the structure of the system they are using to the problem domain and that expose the complex translation and transformation process that occurred as their domain information processing requirements were expressed as a system design and in a system artefact.

Several accounts of the nature of explanation in knowledge-based information systems have highlighted the importance of *justifications* in the systems explanation equation (Gregor & Benbasat, 1999; Ye & Johnson, 1995; Langlotz, & Shortliffe, 1989; Swartout & Smoliar, 1987). Theories of explanation from the philosophy of science are particularly concerned with providing answers to *why* questions by explicating the reasons or causes of an event or event class. What Clancey called *support* knowledge was highlighted as an essential element of any explanation that attempts to relate the structure of a system to its problem solving strategies (relating the *what* to the *why* and *how*). The framework developed here provides an explication of the types of information that might be included in a *why* explanation of an IS artefact. As discussed in the last chapter, very little research has addressed the issue of how the necessary explanatory knowledge for an IS might be obtained. The next chapter will develop one such approach and subsequent chapters will investigate its efficacy empirically.

4 Explaining with Design Rationale

This chapter investigates the field of design rationale as an approach to capturing and representing the information required for explanation of an information system. The perspective is that this approach to information systems design has the potential to provide us with a depth of explanatory content that is not achieved through traditional IS specification methods. Chandrasekaran and Swartout (1991, p. 47) argue that:

> *"...[the] more explicitly we represent the knowledge underlying a system's design, the better its explanations".*

It is argued here that the analysis and design phases of an IS development project are the most productive stages at which to gather the information needed to explain an IS. During these stages, much of the application domain, system structure, and system function knowledge that will determine the system's behaviour in relation to its problem domain is made explicit. The essential question is how to best capitalise on this brief period of exposure and capture the information needed to explain the transformation of problem domain and systems development knowledge into a working IS.

Moran and Carroll (1996) distinguish between the process and product of a given design effort. Typically, when a design effort has been completed, organisations are left with the products of design including the designed artefact and a set of documentation that describes the artefact in highly variable levels of detail and quality. They argue that this documentation rarely includes a complete record of the initial motivation for the project including project team and external stakeholder deliberations and negotiations, and the reasons that were finally applied to justify the inclusion and form of particular system features. It has been argued that to achieve understanding of a system means making transparent the concepts that underlie its architecture (Wenger, 1987). In the design rationale approach, these concepts are captured and expressed as the "argument behind the artefact" (MacLean, et al., 1989).

This first section of this chapter frames design rationale within the milieu of information systems development. The second section provides a review of design rationale's theoretical foundations, the techniques and tools that have been developed

to support the approach, and some of the results reported by researchers and practitioners who have applied the approach in the field and in the laboratory. The final section shows how design rationale relates to the elements of IS explanation identified in the last two chapters and proposes it as mechanism for capturing and representing the deep knowledge behind a system's design, the justification for a system's architecture relative to its domain.

4.1 The Information Systems Development Context

Many modern information systems are large, distributed, and extremely complex, both in their technical architecture (how they work), their functional architecture (what they do), and in how they are integrated with the organisational and social context where they are used. The number and range of factors that might help to explain various aspects of an information system in this context are correspondingly complex. In attempting to develop a framework and an approach for integrating explanation knowledge capture into the information system analysis and design process, consideration must be given to the fact that IS developers and other stakeholders such as managers and project sponsors are unlikely to welcome the introduction of new overheads to the development process. Any new techniques for capturing information generated during the different phases of the software development and implementation lifecycle must be integrated as unobtrusively as possible into the day-to-day activities of the team responsible for the system (Moran & Carroll, 1996).

Information systems are commonly delivered over budget and over schedule; and many cases of high-profile projects resulting in near-complete implementation failure are reported in the literature (e.g., Flowers, 1996; Glass, 1997). Often, specification errors in the analysis and design phases of these projects are identified as the cause of these problems, and they are especially disruptive to organisations as they can lead to extensive rework after the system is already in use (Boehm, 1981). In his seminal paper, Brooks (1987) identified the source of software engineering's increasing costs and repeated failures as:

1. Complexity – software systems are inherently complex and this complexity cannot be abstracted away. The size of a system increases

exponentially relative to complexity so little can be done to control the costs associated with large, complex systems.

2. Conformity - software systems are complex because they need to conform to the *man-made* principles inherent in organisations. Software complexity is not due to the constraints of physical processes that can be reduced to first principles.

3. Changeability – software systems are embedded in a matrix of other systems, hardware, users, organisations, cultures, etc. that change and evolve and thus force change on the software system.

4. Invisibility - because software does not exist in three-dimensional space, it is difficult to visualise. This makes it much more difficult to conceptualise how a given design specification will translate into a working system.

Since Brooks's paper, information systems have become ever larger, more complex, and more pervasive. For example, Cusumano and Selby (1997) report that the Windows 95 operating system now consists of 11 million lines of source code and its development and maintenance involves a team of 200 programmers and testers. In response to the challenges of managing software projects of this scale, Microsoft, among other companies driven by quality and schedule delay problems, has found it necessary to de-emphasize the use of structured development methodologies in favour of more *ad hoc* methods such as their "synch and stabilize" approach. Though some argue, contrary to the received view, that the practice of software engineering is improving and that its products are becoming more reliable and cost effective (Glass, 1996), this de-emphasis on the use of more formal engineering and documentation approaches presents additional obstacles to the introduction of any new explanatory knowledge capture and representation efforts.

Devanbu, et al. (1991), drawing on Brooks's, argue that the essential problem in software understanding is the interaction between complexity and invisibility. While on smaller projects, or in the early stages of larger ones, members of the development team may possess an adequate understanding of the relation between the domain and design rationale of the system being built, as a project evolves and as new project members become involved, this understanding dissipates to critical levels. System

changes made by those who do not understand the original architectural model only exacerbate the problem as the structure of the system drifts further and further from what was once, potentially, a coherent design. In the use context, the problem multiplies as project stakeholders such as users, managers, and the system maintenance and support personnel attempt to understand the system model. Each has access to an increasingly diminished understanding of the original architecture, and the information they do have becomes less relevant as they attempt to apply a system in new and novel ways. Because the system model is in Brooks's sense invisible, efforts to establish or re-establish the level of comprehension once possessed by the original architects are faced with potentially intractable problems of knowledge reconstruction.

One field study of the design process for large systems (Curtis, et al., 1988), which focused on the behavioural dynamics of the software design process, identified the most significant problems facing successful project delivery as:

1. An overall lack of application domain knowledge on project teams, and its concentration in a small number of exceptional individuals.

2. Project requirements that evolve as a project progresses and that often conflict across organisation boundaries.

3. Communication and coordination breakdowns within project teams, between project teams and management, and between project teams and project customers.

The study found that to a large degree project success was dependent on the ability of typically one or two team members to grasp and communicate to other team members the key issues in the problem domain relative to the architecture of the system under construction. In particular:

> *"Their crucial contribution was their ability to map between the behavior required of the application system and the computation structures that implemented this behavior..." (Curtis, et al, 1988, p. 1272)*

These designers were exceptional in their ability to understand and integrate the knowledge of one or more application domains into a coherent system model. They also possess the ability to identify "unstated requirements, constraints, or exception

conditions" that help to map the system model to the problem domain (Curtis, et al., 1988, p. 1272). Because these skills are scarce and take time to develop, once a designer attains this level of knowledge much of their time is spent communicating their vision of domain model and system model mapping to other team members and project stakeholders, rather than applying their knowledge integration skills to new development problems.

Curtis et al. (1988) report that a problem initially attributable to changing project requirements is actually the more complex result of an interplay between the project team slowly coming to understand the problem domain, and customers or users achieving better understanding of their domain requirements relative to the evolving system model. Project documentation approaches were problematic as facilitators of communication within project teams and among external project stakeholders. Written documentation was not seen to fit with the "dialectic" used to resolve discrepancies between evolving project requirements and the evolving system model, and project documentation was seen as time consuming and unwieldy as the scale of a project increased. Though the most effective communication medium identified was informal and inter-personal, one theme that emerged from the study was the importance of documentation approaches that emphasise project *argumentation*, rather than static representation.

The inherent problems of large IS design are resonant of those discussed in Chapter 2 relative to KBS projects, where knowledge acquisition is still reported as one of the major challenges facing KBS developers (Liebowitz, 1997). Clancey (1983) discussed the difficulties in maintaining and extending a knowledge base for which the design is not fully documented. Stefik and Conway (1982) argue against the commonly held notion that knowledge engineering involves access to a small number of domain experts who are able to articulate the content of a relatively stable knowledge space in some domain. Instead, they argue that knowledge engineering must address the distributed and volatile nature of knowledge states in a given *community of practice.* Though the "knowledge-processing" products of research into artificial intelligence are increasingly being packaged into modular components for inclusion in more traditional IS technologies (Hayes-Roth & Jacobstein, 1994), it may well be that the knowledge acquisition and management problems of these large and complex 'traditional' information systems are even greater than those of KBS. While a given

KBS typically makes use of a relatively homogeneous type of data structure (production rules or frame) and processing mechanism (the inference engine); large, distributed, enterprise information systems usually are made up of a range of heterogeneous data structures and processes packaged into application specific modules each with a potentially different set of users.

4.1.1 Remedies?

A plethora of devices have been proposed to provide relief for problems inherent in large IS development projects. These can take the form of a development process, a set of products or deliverables, a tool or tool set to assist with development tasks, or an overriding philosophy to guide developers and other project stakeholders. The reference standard IS development process is still the "waterfall" model (Royce, 1970), which describes the system development process as a series of incremental steps including, at least, analysis, design, code, and test. The "spiral" model proposed a more evolutionary and iterative approach to the systems development lifecycle and explicitly included customer or user checkpoints as part of its cycle (Boehm, 1988). Parnas and Clements (1986) argue that despite the problems inherent if attempting to prescribe a software design process, it still pays to present the design as the result of a rational process because:

1. When designers do not know how to proceed, a guideline can provide them with ideas.
2. Attempting to follow an idealised process for example, gathering 'all' requirements up front, will lead to a better product than simply admitting defeat and following an ad hoc process.
3. A rational design process can be shared across an organisation. This helps people to understand where a project is and what is required for a given phase.
4. Following a process makes it easier to measure progress.
5. Project review is easier when a process is followed.

Product-oriented approaches to IS development focus on a set of prescribed deliverables that typically correspond to project milestones or iterations. For example, the waterfall process model is usually associated with structured analysis and design (DeMarco, 1979; Yourdon, 1989) and its many variants (see Pressman, 2001; Avgerou & Cornford, 1998; Avison & Fitzgerald, 1995), which focus on procedural and data decomposition into successively more detailed data flow diagrams (DFDs) and entity-relationship models (ERMs) respectively. The unified modelling language, or UML (Booch, et al., 1998), is an amalgam of several different object-oriented analysis and design approaches that prescribe use case diagrams, class diagrams, object-sequence diagrams and others as part of the more holistic behavioural decomposition common to object-oriented programming. Prototyping in its many forms is a product-oriented approach that focuses on putting working software in the hands of users or customers and evolving the product based on their feedback (see Tanik & Yeh, 1989). Other product approaches include formal methods such as Z (see Spivey, 1989) and hybrids of formal and semi-formal methods such as cleanroom software engineering (see Mills, et al., 1987; Poore & Mills., 1989).

Tool-oriented approaches to an IS development project include computer-aided software engineering (CASE), software information systems (SIS) in many different forms, and environments to facilitate computer-supported collaborative work (CSCW). CASE is an umbrella term that typically describes a range of software development tools supporting analysis and design, construction, validation and verification, project management, and documentation, all organised within a structured, integrated environment (Pressman, 2001; see also Vessey & Sravanapudi, 1995). SIS include knowledge-based systems that attempt reason about the capabilities of software components in support of component reuse and validation of overall achitectures (e.g., Harandi, 1988). CSCW, another umbrella term, is both a family of software tools known as group support systems (GSS) (see Bidgoli, 1996), and a "worldview" that emphasises a collective approach to information management and work (Kling, 1991). The literature on CSCW, GSS and related tools is large, see (Grudin, 1991; Kling, 1991) for overviews and (Baecker, 1993) for a collection of seminal papers.

Processes, products, and tools such as those described above may be interleaved within an overriding philosophy of IS development. These range from the traditional

systems approach described by the waterfall and spiral models to those that more explicitly account for the human and social dimensions of the IS development process. While the former find their expression in cleanroom and other formal software engineering methods, the latter include different modes of participatory design (e.g., Mumford, 1996), rapid application development (e.g., Kerr & Hunter, 1994), and most recently, so-called extreme programming (Beck, 2000). Though each of these approaches has committed adherents and all have met with some measure of success in different organisations and project development contexts, none have garnered anything approaching universal acceptance in the systems development community.

Despite the dissemination of these methodologies, techniques, tools, and philosophies, IS development as practiced is often comprised of *ad hoc* activities rather than strict adherence to a particular prescribed approach (Whitley; 1998; Introna & Whitley, 1997; Turner, 1987; Rosson, et al., 1988). Practising designers are seen as suspicious of those who would analyse their intuitions (Alexander, 1964) and hostile to attempts to formalise these activities (Cross, 1984). For many, the central problem inherent in large development projects is seen to be largely a function of coordination, in particular inter-personal coordination (Kraut & Streeter, 1995; Brooks, 1975). Kraut and Streeter define coordination in the software development context as:

> *"...it means that different people working on a common project agree to a common definition of what they are building, share information, and mesh their activities." (Kraut & Streeter, 1995, p. 69)*

Though research suggests that informal, interpersonal communication is perhaps the most used and most effective method of achieving coordination, the costs of this kind of coordination in terms of person-hours and the ephemeral nature of the project knowledge generated this way are well documented (Brooks, 1975).

4.2 What is Design Rationale?

Design rationale differs from 'mainstream' information system analysis and design approaches in several respects. First, in its theoretical orientation it explicitly acknowledges the "wickedness" of many large and complex development projects (Rittel & Webber, 1973). These problems typically lack an optimal solution and their complex nature forces a series of "satisficing" choices so that working systems can be

delivered within practical constraints (Simon, 1996). Second, as an approach DR is largely unprescriptive, focusing instead on capturing what designers *do*, and enabling a *reflective* element to these tasks (Schön, 1983). Cross (1984) claims that the design 'movement' has progressed through four distinct phases that he identifies as prescription, description, observation, and reflection. Design rationale may be characterised as an outcome of this fourth phase. Finally, design rationale is concerned with the entire *design space* of a project rather than exclusively with the artefacts that emerge from separate design activities (MacLean & McKerlie, 1995; MacLean, et al., 1996). The arguments and deliberations that occur within this design space and the justifications that underlie the form and function of the artefacts that emerge from it are of central importance. Design rationale is not a stand-alone solution to the system specification problem. DR is typically used with other project artefacts, methodologies, tools, and development philosophies (MacLean & McKerlie, 1995; Potts, 1996).

Information systems have been characterised as "frozen organisational discourse" (Bowker & Leigh-Star, 1994), in the sense that they embody an organisation's perspective of what is required to manage information in a particular problem domain, in a particular way, at a particular point in time. With its focus in the entire design space, design rationale is an approach to capturing this discourse and the range of factors that ultimately impact a design including, especially, "how it might otherwise be" (MacLean, et al., 1996, p. 55).

4.2.1 Theoretical Foundations

Bush (1945) provides a succinct account of the motivation and goals of design rationale and, more generally, knowledge management:

> *"Presumably man's spirit should be elevated if he can better review his shady past and analyze more completely and objectively his present problems. He has built a civilization so complex that he needs to mechanize more fully if he is to push his experiment to its logical conclusion and not merely become bogged down part way there by overtaxing his limited memory. His excursion may be more enjoyable if he can reacquire the privilege of forgetting the manifold things he does not need to have immediately at hand, with some assurance that he can find them again if they prove important."*

Moran and Carroll (1996) claim the work most frequently cited as the theoretical or conceptual basis of design rationale is that of Alexander (1964); Rittel and Webber (1973); Simon (1969); and Schön (1983).

Alexander

Christopher Alexander's (Alexander, 1964; Alexander, et al., 1977) work on design theories in architecture revolve around two basic themes. First, design problems are too complex to be managed without the support of a structured approach and associated techniques and tools. Alexander was among the first to apply rational, mathematical processes to architectural design, though he acknowledges that designers are generally deeply suspicious of any methodology that might constrain their intuition (Alexander, 1964). Second, the practice of design should be cumulative, designers in a given domain continually struggle with a basic set of problems that can be described by reusable solution "patterns". Alexander argued that these patterns could be exploited by creating generalised abstractions that could be applied in analogous situations. As mentioned earlier in this chapter, this latter point has been highly influential in the development of design pattern languages for object-oriented software development and reuse, which explicitly include a partial rationale in their specification (Gamma, et al., 1995).

Rittel & Webber

Rittel and Webber (1973) provide a critique of planning and problem solving in the domain of urban and regional development and government that suggests an alternative approach to the issues discussed in the previous sections on IS projects. They recognised that complex planning and development projects often exist in a rich web of social and political forces that make them incompatible with the methods commonly employed in formal engineering. The network of factors that influence the nature of these problems is immense and their relative influence variable making each instance unique. These problems are characterised by being poorly defined and, at some level, ultimately undefinable. This lack of definition makes identification of a "stopping rule" to differentiate between complete and incomplete, and sometimes even good and bad solutions extremely difficult and in some cases impossible. Additionally, problems of this sort typically have consequences outside of the group

charged with their 'solution', they generally need to operate effectively on the first try, and someone or some group is always accountable if they fail.

Faced with these "wicked' problems, Rittel proposed an approach to their management (since they can never be truly solved), the issue-based information system (IBIS) is one of the foundations of the problem representation approaches upon which design rationale is based. IBIS is outlined and discussed later in this chapter.

Simon

Simon (1996) provides a structured, cognitive account of the design problem that views design as a search through a problem space. Though he outlines a range of mathematical and statistical techniques that be used to search the problem space in constrained domains, he acknowledges that design space search is often one that seeks a "better" rather than "best" solution. Simon's theory of "bounded rationality" suggests that ultimately our cognitive apparatus will act as a bottleneck in comprehending the number and range of variables that impact the efficacy of a given design. He argued against the idea that humans are able to apply the equivalent of a comprehensive, rational utility function when faced with these complex problems. Instead, he invented the term "satisficing" to describe scenarios where humans problem solve by:

> *"...looking for alternatives in such a way that we can generally find an acceptable one after only moderate search." (Simon, 1996, p. 120)*

As an approach, DR makes no claim respective of its ability to improve the situation described by Simon. What it does claim is that better problem structuring methods can support and potentially improve our ability to consider the information that *is* available for design decisions. As originally proposed by Engelbart (1963), embedding such methods in electronic information processing tools can only enhance this effect.

Schön

Donald Schön's (1983) highly influential critique of the professions and the status of professional knowledge identified the growing "crisis of confidence" being experienced by both the lay public and the professions themselves (e.g., medicine,

law, engineering and technology, among others) relative to claims of special and invaluable knowledge in these fields. Schön claimed that advances in areas such as medicine and space exploration were offset by professionals' failures, such as the continued occurrence of war, poverty, and environmental pollution. The situation Schön described was echoed earlier in this chapter where it was shown that, despite the obvious successes of information technologists and their near-celebrity profile in the internet-obsessed media, significant problems still pervade the development of consistently reliable information systems.

Schön argued that, to a great extent, the problems being experienced by the professions are the result of a disjunction between their highly specialised but narrow training, and the complex, unique, and socially embedded "messiness" of the problems they encounter in practice. He identifies two phenomena that are central to our ability to deal with these kinds of problems but that are not part of the professional curriculum, these are *knowing-in-action* and *reflection-in-action.* According to Schön, an expert professional's normal operating mode is based on knowing-in-action where:

> *"Often we cannot say what we know. When we try to describe it we find ourselves at a loss, or we produce descriptions that are obviously inappropriate. Our knowing is ordinarily tacit, implicit in our patterns of action and in our feel for the stuff with which we are dealing. It seems right to say that our knowing is in our action." (Schön, 1983, p. 49)*

However, situations such as encountering a particularly interesting or difficult problem, or surprise at a particular outcome result in a change of mode from knowing-in-action to reflection-in-action. In these situations:

> *"As he tries to make sense of it, he also reflects on the understandings which have been implicit in his action, understandings which he surfaces, criticizes, restructures, and embodies in further action." (Schön, 1983, p. 50)*

Moran and Carroll (1996) ask whether there are "natural points of reflection" in a systems development project, for example, status meetings, prototype reviews, formal checkpoints, etc., which can be used as cues to generate design rationales. Clancey (1983) argues that constant reflection on the structure and content of any knowledge base is essential to the development of a framework to support explanation in a given domain. The tension is that in many cases, humans will only enumerate the exact

steps that led to a particular decision if pressured to do so (Suchman, 1987). The remainder of this chapter will explore how design rationale is commonly used in support of reflection-in-action, and the costs and benefits that accrue when the approach is applied.

4.2.2 Techniques & Tools

A range of representation strategies and associated tools has been developed to support DR in action. Though they all follow similar typed node-typed link structures (Shipman & McCall, 1997), these formalisms vary in terms of how structured they are with the acknowledged trade-off being increased tractability by users versus increased tractability by machines (Lee & Lai, 1996). Lee (1997) identifies four different approaches to capturing design rationale: reconstruction; as a by-product of the development methodology; through the use of an integrated "design apprentice"; and by automatic generation. Reconstruction involves the production of design rationales using records from the design process including notes, specifications, audio and videotapes, e-mails, and other materials. A more cost-effective method is for the design rationale to be produced as a natural, integral by-product of the design process and design techniques that are employed. In the apprentice approach, the design rationale technique is integrated into the design (CASE) tool that is used by the development team. The design process and activities supported by the tool include capturing the design rationale information. Finally, automatic design rationale generation tools capture the decision rules applied in a design problem and construct design rationale representations from this information.

This section reviews a representative selection of the formalisms that have been developed explicitly to support the design rationale technique. These range from relatively unstructured approaches to design rationale as formal engineering models. For a more detailed account of DR's roots in representational strategies, see Shum (1991).

Toulmin

Though not a design rationale formalism *per se*, Toulmin's (1958) approach to argumentation as a form of logical reasoning is reviewed here given its role as a prototype for DR languages. Toulmin developed an alternative to theories of formal

logic that attempts to provide a more practice-based approach to the analysis of problem structures. His approach involves the production of argument structures that are represented graphically as in the following example, which shows an argument around how to handle multiple currencies in a sales capture IS.

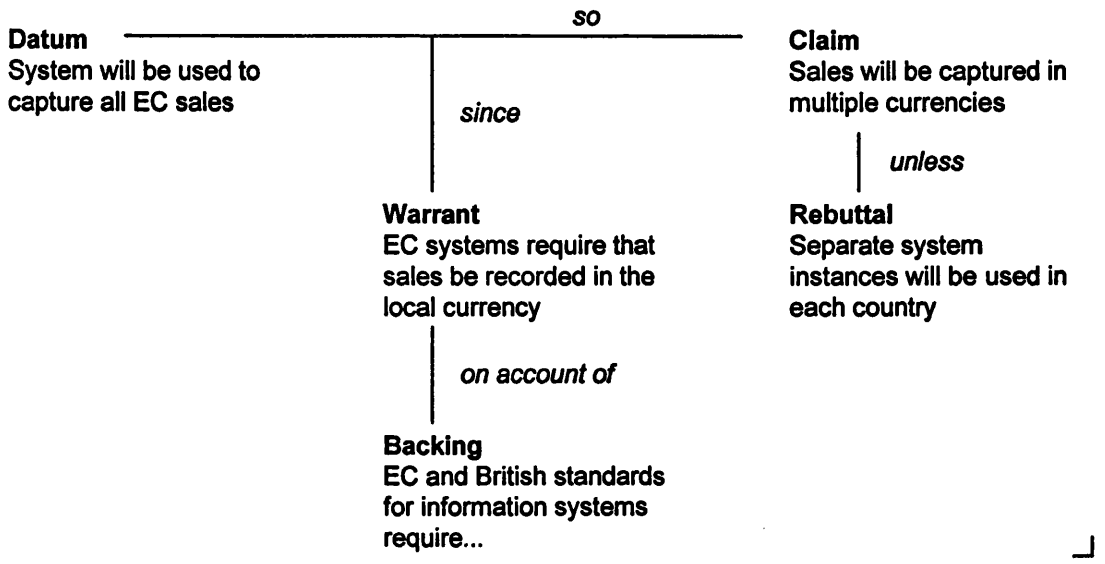


Figure 7 - A Toulmin Argument Structure

The Toulmin model consists of five object types and four relationship types. A *claim* is the initial assertion that we are attempting to prove. A *datum* represents the key fact that is given to establish the claim, *system will be used to capture all EC sales* therefore *sales are captured in multiple currencies*. *Warrants* are additional foundations upon which the argument is based. *Backing* are laws, standards, norms, that support the warrants. Finally, *rebuttals* are arguments against the central claim. The Toulmin model of structured argumentation is considered the logical and notational basis for modern design rationale approaches. An analysis of the foundations of Toulmin's and other argument representation formalisms can be found in Hair & Lewis (1990).

IBIS

According to Moran and Carroll (1996), the earliest proposed design rationale method was Rittel's IBIS, the Issue Based Information System (Rittel, 1984). Rittel's concern was with what he called the "wicked" problems of architectural design and city planning, problems that consisted of hundreds or thousands of different issues and that involved large teams of stakeholders attempting to develop solutions. The IBIS

model takes an argumentation view of the planning and design process and identifies three key elements that make up a given debate. Elements of a problem domain are represented by three node types: *issues* are identified for which stakeholders take *positions*; these *positions* are backed by *arguments*.

In addition to these three central IBIS objects, the model includes eight different link types that may be used to express the relationships between objects. These include *generalizes*, *specializes*, *replaces*, *questions*, *is-suggested-by*, *responds-to*, *supports*, and *objects-to*. The following diagram shows the allowed object and relation types in IBIS, with the addition of the *other* node and link types included in the gIBIS model discussed shortly (from Conklin & Begeman, 1988):

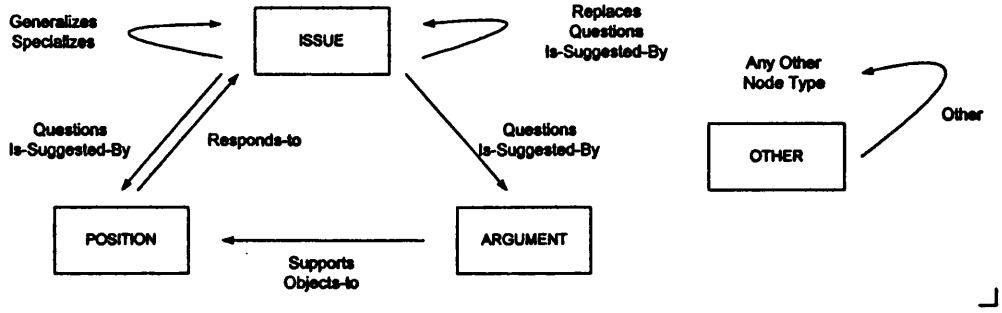


Figure 8 - The Issue Based Information System (IBIS)

Conklin and Begeman (1988) argue that the content of collaborative work may be characterised as consisting of three types of utterance: *substantive*, the actual content under discussion; *annotative*, higher level comments *about* the substantive issues; procedural, how the work is being performed. The relatively unstructured nature of IBIS and the weak semantics of its node and link types means that its direct descendents are limited in their ability to represent these non-substantive issues, or meta issues. Other problems with IBIS that have limited its further development include: complexity of the representation (relative to unstructured note-taking); lack of support for *identifying* issues; and lack of support for clarifying central versus peripheral issues (Shum, 1991). The notation's lack of representational specificity has been addressed in some of the IBIS descendents described below.

gIBIS

The gIBIS project (Conklin & Begeman, 1988) is an IBIS and hypertext-based DR tool with graphical extensions. gIBIS extends the basic IBIS model with the addition

of a "catch-all" node and link type, *other*, which may be used to describe elements of a deliberation that might at first be ambiguous. The gIBIS tool is based on a relational DBMS and is designed specifically to manage the large volume of design information that arises on larger, more complex projects. gIBIS supports both graphical and text views on the node-link structure of a given DR. A limitation of the gIBIS tool, and the IBIS notation generally is its lack of support for a *goal* operator to represent requirements as well as links to design artefacts such as specifications and other documents.

Procedural Hierarchy of Issues (PHI)

The Procedural Hierarchy of Issues (PHI) another IBIS-based DR formalism (Fischer, et al., 1996) attempts to solve a number of key deficiencies in the original IBIS framework. One such issue is that IBIS has no representation for the dependencies between different design issues, they introduce a link type called *serve* to designate those relationships where the resolution of one issue affects the resolution of another. A second is that IBIS does not support capture of design issues that are not deliberated, Fischer et al. redefine the IBIS issue concept to include any design question that arises during deliberations. The PHI approach has been implemented in several software support tools including MIKROPLIS (McCall, et al, 1981, cited in Fischer, et al., 1996), one of the first hypertext-based issue deliberation systems.

QOC

The semi-formal notation *QOC* for *Questions*, *Options*, *Criteria*, is part of the *Design Space Analysis* (DSA) approach developed by MacLean and others at Rank Xerox EuroPARC (MacLean, et al, 1996; MacLean & McKerlie, 1995). DSA is concerned with looking beyond the artefact, the specifications and resulting artefact, produced by the design process to the broader issues that resulted in its development, the design space. A DSA is a separate deliverable meant to be produced alongside the designed artefact and other supporting specifications and documentation. DSA/QOC contributes to the design process by exposing assumptions being made by designers, raising new questions, challenging the legitimacy of the design criteria, and showing how newly identified options might overcome problems with those previously identified (MacLean, et al, 1996). This DSA/QOC notation allows designers to capture not only elements of the final design, but also the reasons why a final design

turned out the way it did, what alternatives were available, and why a particular one was chosen (MacLean et al., 1989).

In the QOC notation, *questions* highlight issues that have been identified as relevant to the design, *options* are the potential solution approaches that have been identified to address a given question, and *criteria* are the reasons that are considered for or against each of the identified options. Whether a criterion is considered a positive or negative factor in the evaluation of a given option is represented in the links, known as *assessments*, between options and criteria. Supporting criteria are linked to options using a solid line; Criteria that weigh against a given option are linked using a dashed or dotted line. One important note is that assessments in QOC are not assigned weights to represent their relative importance to the argument for an option. QOC's creators argue that while such "levels of assessment" are desirable, the cost in additional complexity in capture and use does not outweigh the benefits (MacLean, et al., 1996).

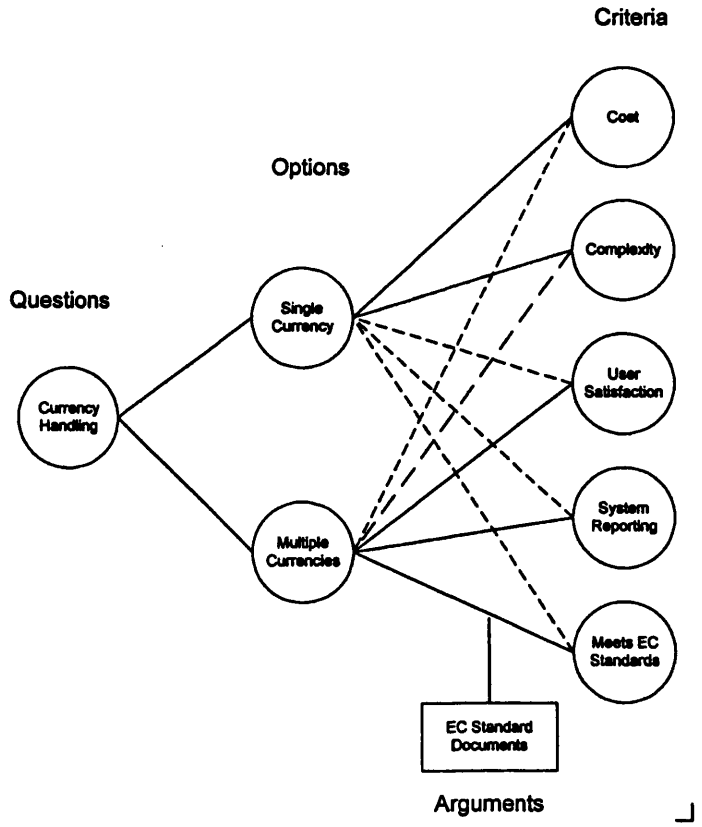


Figure 9 - Questions, Options, Criteria (QOC)

The figure above presents a simple QOC example showing an assessment of a design question related to the currency-handling model of a business software application.

In addition to the basic vocabulary, *arguments* may be linked to an arc to provide additional information on the relationship between a criterion and an option. Arguments can take the form of well established theories, empirical evidence, or other data that justify an assessment. Another feature of QOC is the use of *bridging criteria* that represent how a general design requirement or goal, for example, ease of use, relates to a specific design option (MacLean, et al., 1996). Finally, Options may themselves give rise to new Questions each with their own local DSA.

Though QOC is a descendent of the IBIS method, its developers contrast the approaches across several important dimensions (MacLean, et al., 1996). IBIS includes Issues, Positions, and Arguments as its basic operators, however, these constructs are generic, they can refer to any issue raised in a design discussion as opposed to QOC's, Questions and Options that specifically relate to design questions. IBIS Arguments are meant to be captured 'on the fly' and are less structured than QOC which is designed to guide as well as record design deliberations. In IBIS, Arguments may include embedded criteria as opposed to the explicit representation of criteria in QOC. This explicit representation of design criteria, rather than more generic arguments in IBIS, has been found useful on at least one engineering project (Karsenty, 1996). Finally, there is a trade-off between ease of construction (IBIS) versus ease of use (QOC) based on the less structured approach advocated in the former approach. MacLean, et al. (1996) emphasise the complementary natures of these two method and suggest an approach to their integration that involves the use of IBIS in real-time time to capture design arguments as they emerge. The IBIS database may then be used as an input to the more structured QOC, where a reflective view of the design discussion can lead to more careful analysis.

Research on applying the QOC formalism and the DSA approach in practice suggests that over 90% of design deliberations can be captured using the method (MacLean & McKerlie, 1995). However, it has been noted that the process of classifying design utterances into the QOC framework is not always straightforward (Buckingham Shum, et al., 1997; MacLean, et al., 1996). Further research into the cognitive

usability of DSA/QOC can be found in (Buckingham Shum, et al., 1997; Shum, 1991) and is included in parts of the sections that follow.

DRL

The Decision Representation Language, or DRL (Lee & Lai, 1996) represents a shift away from QOC towards the more structured end of the DR representation spectrum. DRL is a structured, rigorous approach to design rationale capture and representation that, its creator claims, overcomes many of the expressive inadequacies of other DR notations. Lee (1990) argues that by being rewarded with increased utility when the DRL data is used, users are more likely to be motivated to capture the information. DRL's vocabulary includes a rich set of objects and relations that are more differentiated and strongly typed than those found in either IBIS or QOC. DRL is designed to be extensible, new object and link types may be added based on additional concepts that are introduced in a given domain.

The basic DRL vocabulary consists of the following objects and relations.

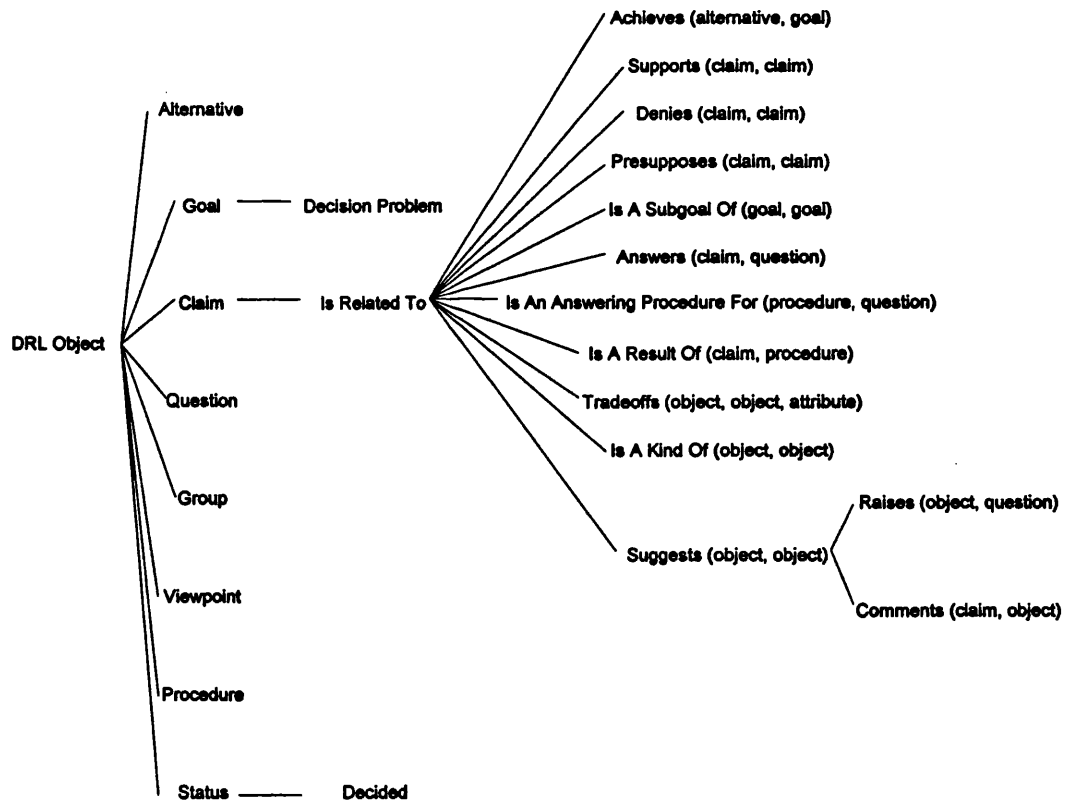


Figure 10 - Decision Representation Language (DRL)

The core node types in DRL include *Decision Problems*, which represent design issues; they correspond to the object type *Questions* in the QOC framework. *Alternatives* are the options being considered for a particular decision problem. Alternatives are considered relative to a particular *Goal*, which represents a target or advantageous state. The *Is Related To* object and all of its subtypes are used to link objects in a relationship. The parameters to these relations, the types of object that are allowed for a given relation type, are shown in brackets after the link type name. Other object types such as *Question*, *Group*, and *Viewpoint* are considered auxiliary types for qualifying or elaborating on other objects in the framework.

One of the more important elements of DRL is the addition of *Goals* as a node type. Lee (1990) argues that goals are important as they allow design questions and alternatives to be considered relative to the overall objective or motivation for a decision problem.

Among the incremental benefits gained by using a more expressive design rational formalism such as DRL is that it allows the capture of both the logical structure, why things are the way they are, and the historical structure or why features replaced or modified other features over time (Lee & Lai, 1996). However, in contrasting QOC to the significantly more structured approach taken in the DRL framework, MacLean, et al., (1996) acknowledge that while DRL's rich set of operators might form the basis for any extensions to QOC, the addition of new DR language elements should be carefully considered due to the decreased ease of use and ease of learning that may result as notations become more complex.

SIBYL

SIBYL (Lee, 1990) is a DR system that implements the DRL formalism described in the last section into a group design decision support tool. The system designed to support qualitative design decision making and knowledge sharing. SIBYL includes both a graphical decision tree display and a decision matrix form for representing decision problems. SIBYL and the underlying DRL formalism are designed to extend the gIBIS/IBIS model so that DR information may be treated as a knowledge-base that can provide a range of automated and semi-automated services such as design dependency management. No field studies reporting on the utility of the DRL/SIBYL approach have been reported.

Other DR Approaches

A number of research projects have or are investigating the utility of design rationale approaches in support of formal engineering (KSL, 2000; Gruber & Russell, 1996; Klein, 1993). The 'How Things Work' project at Stanford University's Knowledge Systems Laboratory (KSL, 2000) is concerned with modelling, simulating, documenting, and explaining complex devices. The approach to DR employed on this project relies on the pre-existence of formal engineering models that specify the device under analysis (Gruber & Russell, 1996). The Design Rationale Capture System (DRCS) described in Klein (1993) is designed to support collaborative capture of formal DR across concurrent, distributed design teams.

Gruber and Russell (1996) argue that DR is often constructed and inferred from stored information rather than stored as complete answers to designers questions. They characterise explanations based on this type of automated reasoning and strongly typed data as strong "explanations". Their Device Modeling Environment (DME), relies on formal ontologies and behavioural specifications to produce model-driven explanations of the device in question.

Another detailed coding scheme attempts to account for the rich dialectic that takes place during the software design process (Elam, et al., 1991). This coding scheme is organised around two major dimensions, the behavioural dimension and the content dimension. The behavioural dimension consists of several types describing design deliberations: expository, communication of a mental model, problem, or issue; acquisitive/facilitative, attempts to reconcile understanding among different group members; and procedural, the management of the meeting and project. The second dimension is the topic dimension, which consists of types such as: goals, design approach, project management, and system requirements. Each dimension may then be attributed using additional operators to describe the content of the utterance. The idea behind this approach is that each unit or utterance of the design meeting is assigned a unit number, a behavioural dimension code, a topic dimension code, a speaker identifier, and a reference number. The reference numbers refer to other units upon which the current unit is based. Though the utility of design information captured at this level of detail is possibly great, it is unclear how this information would be captured in the heat of a real-world design meeting.

Each of these more formal and more detailed approaches is predicated on the argument that the benefits of storing strongly typed design information that can be used for deductive inferences may outweigh the costs of capturing and structuring this information when the device has a high failure cost or a safety critical element, for example, an aircraft. Most information systems would not meet the criteria that support capturing and structuring design data to the level of detail that they require. However, some of these formal approaches (e.g., Gruber & Russell, 1996), also include the notion of 'weak' explanations from design rationale information that has not yet been strongly typed. They argue in fact for a highly unstructured, unprescriptive approach to capturing DR for what they call 'weak' explanations, design rationale users should be given access to deliberations in an unstructured format and allowed to construct explanations themselves from this information.

Several propose approaches to DR that acknowledge the essentially incremental nature of the design development and design process. Shipman and McCall (1997) suggest that DR is best captured as unstructured deliberations, and then incrementally structured over time as needed. Part of the project they describe is the development of two software tools, PHIDIAS and Hyper-Object Substrate (Shipman & McCall, 1997), based in the PHI formalism discussed above, to support the incremental formalisation approach they prescribe. Hyper-Object Substrate (HOS) is focused on the capture of design deliberations in a multitude of formats, for example e-mail and USENET news files, for later incorporation in the design analysis. PHIDIAS relates the DR to design artefacts and provides design critiquing services through the use of software agents. Fischer, et al. (1996) also employ the PHI method that in attempting to account for the feedback mechanism that exists between design and construction.

Several DR tools have been developed to support capture of rich, multi-media design meeting content. The *Raison d'Etre* prototype (Carroll, et al. 1994) provides access to a data base of video clips from design meetings. The purpose of this system is to capture aspects of the community that grows out of interactions on a design project, the interaction of the formal and informal, and the technical and social. These clips may later be used by team members to understand the culture of their design community, and to support students and researchers attempting to learn about how technical issues are addressed within the social dimension of design.

The REMAP system ("representation and maintenance of process knowledge") uses a semiformal representation to manage the design and maintenance of large-scale software systems (Ramesh & Dhar, 1994). Their approach is based on the IBIS approach, however, their conceptual model extends the issues, positions and arguments from IBIS to include *requirements* as the basic input and decisions as the basic output. Among their goals is the development of a concept classification framework to help structure the output from informal design meetings. REMAP/MM (Ramesh & Sengupta, 1995) is an IBIS-based design rationale system that captures design deliberations represented in multiple media including audio and video formats. These representations typically must be analysed and re-represented to be tractable by a Dr system. However, such approaches allow the full depth of the design discussion to be analysed later. REMAP/MM is specifically tailored for use in the systems development context. For example, it includes design objects, the artefacts that result from the design process, in its ontology.

A range of collaborative environments to support social creativity are described in Fischer (1999). A complete bibliography of both commercial and research tools that have been developed to support DR is available at (Buckingham Shum, 2000).

4.2.3 Using Design Rationale

Among the uses and beneficiaries of design rationale most often cited are system designers for better designs, system maintainers for better maintenance, and novice designers for learning about how a particular system was designed as well as the decision-making process followed by experienced designers (Lee, 1997). Design rationale also serves as a documentation base to facilitate communication among the various actors with interests in the design process and the resulting artefact (MacLean, et al., 1996; Shipman & MacCall; 1997). Moran and Carroll (1996) argue that design rationale can be usefully applied in all phases of the system development lifecycle, and is especially useful in the maintenance phase where they believe some of design rationale's most significant cost savings can occur. The following sections will review how some of these contributions are made, section 4.3 will focus on how DR might serve to provide explanations of the designed artefact.

Design Rationale Uses

A design rationale approach true to its theoretical roots in the structured management of "wicked problems" (Rittel & Webber, 11973) through "reflection-in-action" (Schön, 1983) involves integrating a DR formalism and supporting tools into the design process. However, much of the research into capturing design rationale acknowledges the potentially disruptive effects introduced into the process by the need to capture formalised knowledge while attempting to work naturally through difficult issues (Shipman & McCall, 1997; Gruber & Russell, 1996; Buckingham Shum & Hammond, 1994). Several researchers propose reconstructive DR, building design arguments from materials captured in meetings, notes, e-mail, and through other communications channels, as a way to avoid this disruption (Fischer, et al., 1996; Shipman & McCall, 1997; McKerlie, et al., 1993; Shum, et al., 1993). The following sections highlight some of the benefits reported from the use of DR as well as the costs incurred by attempts to structure and capture design deliberations.

Supporting the Design Process

According to Buckingham Shum, et al., (1997), DR may be applied to two different classes of design scenario, which they characterise as *evaluation* and *elaboration*.

Whereas on problems classed as more routine design *evaluations* DR formalisms were found to be more disruptive to the design process, the structure imposed by, in this case QOC, was seen to assist in the *elaboration* of more complex design questions. On these elaboration problems, the benefits of DR approaches include the ability to expose assumptions that have been made in the design, raise new questions concerning the viability of particular options, challenge criteria that have been applied in the selection of a given option, and help to identify new options that may overcome problems with an existing one (MacLean, et al., 1996).

Design rationale is concerned with the effectiveness and correctness of design. According to Gruber (1991, (p. 70), understanding a design's rationale involves understanding: the artifact's structure, the reasons for choosing particular components or implementation approaches, the assumptions about the context in which the device will be used, and any experiences with designing similar artefacts (e.g., design case histories). As Guindon (1990) points out, the most expensive software errors to correct are those that occur at the highest levels of design, and this is the point at which a design rationale is most likely to uncover them. DR proponents claim that it provides verifiable representation of the design (MacLean, et al, 1989). A foundation of DR is its ability to help designers reflect upon their work and uncover discrepancies and errors in their reasoning (Ramesh & Sengupta, 1995).

Elam et al., (1991) cite several studies which show that designers problem-solve by building mental models of the problem space and then applying mental simulations to test possible solutions. These simulations often break down because of cognitive limitations and/or missing knowledge. Because large software projects involve multiple designers, these simulations are also disconnected as different members of the design team have different knowledge in their models. Breakdowns also occur because of social dynamics in group communications, differing assumptions among group members, or more forceful team members dominating the focus of discussion. It has been argued that DR approaches can support recovery from the design "breakdowns" that occur when designers or other stakeholders encounter novel scenarios, those of extreme complexity, or those with special importance (Fischer, 1999).

MacLean, et al. (1995, p. 81) cite well-known psychological theories, as well as their own empirical work, which suggest that designers focus on evidence that supports their initial biases. In the QOC framework, this results in lack of generation of new positive criteria for competing design options and new negative criteria for the options to which they are biased. However, QOC and other DR approaches help to focus designers' and external project stakeholders' attention on where such biases may be having a negative impact on the quality of the design.

DR can support the identification of synthetic solutions which emerge on close analysis of the options available and the criteria that are being applied (Conklin & Begeman, 1988). Bellotti (1993) reports on an effort to analyse and synthesise different HCI design approaches and relate the synthesis to HCI design practice using QOC/DSA. This approach helped to identify the most salient elements of the different HCI theories, develop generalised criteria for HCI design, and relate these criteria to HCI design practice. MacLean & McKerlie (1995) also use DSA/QOC to analyse design options relative to criteria and to synthesise existing options into new ones that satisfy most or all of the identified criteria. These results highlight a sort of 'structured creativity' that can result from the use of DR to reflect upon how elements of the design space interact.

Fischer (1999) argues that DR approaches and tools must include not only the design deliberations but also the artefacts that result from those deliberations. By providing links or otherwise referencing these artefacts, DR can be used to more closely manage their evolution and the interplay between experimental results that result from construction and design deliberations (Fischer, et al., 1996). One approach to addressing the software crisis discussed earlier in this chapter has been through the development of better means of modularising, sharing, and reusing software components and software specifications.

In Potts, et al. (1994), it is shown how DR can be applied to requirements tracing, resolution, and realisation. According to Jagodzinski and Holmes (1989), requirements analysis needs to be performed early with an eye towards the extra dimension of system misfit that may arise if the context of the system is not considered carefully. MacLean & McKerlie (1995) use DR techniques to relate design criteria (in the DSA/QOC formalism) to use scenarios of tasks in the problem domain.

In essence, a criterion is derived from the task support requirements and is used in the analysis of design options relative to this criteria (i.e., the extent to which a given option supports the task). For more complex use scenarios, MacLean and McKerlie propose the use of the two scenario types discussed perviously: envisioner and evaluator. Envisioner scenarios are high level, complex design problems such as "how do we access information". DR may be used to explore (envision) the design space of these meta-scenarios, identify crucial lower-level design alternatives and issues, and then feed directly into more detailed DR representations of evaluator scenarios. Evaluator scenarios are concrete tasks represented in the DR, for example "drop down menus or icons to represent different information access categories". In this case Dr is used to evaluate specific design options relative to concrete use scenarios.

Many researchers argue that 'new' designs are simply variations of solutions that have been applied before (Gamma et al., 1995; Schank, 1986). Researchers into design rationale (MacLean, et.al, 1989), object-oriented system design (Alger and Goldstein, 1992) and artificial intelligence researchers (e.g., Schank, 1986) argue that more often than not, design problems - indeed problems in general - are variations of problems that have already been solved. Although design is typically considered a creative act, the increased use of software application frameworks and component-based software engineering suggest that this creativity is becoming an issue of creative *assembly* of reusable components. Guindon (1990) argues that expert designers perform better than novices because they have a) more design 'structures' on which they can draw and b) the heuristics which are used to apply them.

Shareable software components represent discrete units of functionality that minimise reliance on other components in given system. Modularisation reaches the current state-of-the-art in object-oriented software, where both the attributes (data) and the behaviours (methods, or code) are encapsulated in a programming structure called an object. Objects "expose" only that functionality required to use the component. Though Stefik and Conway (1982) argue that knowledge engineering approaches that attempt to capture the design principles inherent in a community of design practitioners can result in reusable components, significant problems remain with identifying, retrieving, and understanding the semantics of complex software. Though software information systems attempt to provide more comprehensive knowledge of

what a component is and how it performs its task, they typically do not address questions of why the component works the way it does (Devanbu, et al., 1991).

One approach to managing the software design, redesign, and reuse problem that is currently receiving much attention the use of *design patterns* as originally described in the so-called "Gang of Four" book (Gamma, et al., 1995). Drawing on work of a similar nature in the field of architecture (Alexander, 1964; Alexander, et al., 1977), the use of design patterns in software development is the explicit acknowledgement that expert designers are expert because of their ability to reuse designs, but that even experts are seldom able to avoid "design déjà vu" on projects that are new but involve problem analogous to ones they have faced in the past. Design patterns take the form of reusable software specifications with the following elements:

1. Pattern name – provides a common vocabulary that designers can use when discussing alternatives. Assigning good names that are evocative of the patterns function has proven difficult.

2. Pattern problem – describes when to apply the pattern and describes the context of use. Pattern specifications sometime include a list of criteria that must be met in order to apply the design.

3. Pattern solution – the elements that make up the design pattern: relationships, responsibilities, and collaborations.

4. Pattern consequences – the outcomes and trade-offs involved in applying a particular pattern in a particular use context.

Design patterns typically include examples of the pattern in one or more common programming languages. One of the challenges faced in the design pattern project is to develop designs specific to the problem at hand, but also general enough to be used in a wide range of contexts with minimal redesign. The current popularity of the design patterns approach to software development offers a way in which DR can improve its cost-benefit position. One problem faced in the use of design patterns is identification and selection of a pattern for a particular use context and in understanding the consequences of pattern use. Though the design pattern specification includes information to assist in this task, augmenting the pattern with its design rationale may provide additional support for design pattern selection.

Bose (1998) describes a prototype system and approach to capturing DR in support of component-based software architectures. DR is seen to support the evolution of a system as new components are added to the architecture. DR supports the understanding of dependencies that exist in the architectures and that might be violated as new components are added. Software design errors are especially insidious as their effects, sometimes hidden, tend to propagate from one model to another as a system is assembled (Harandi, 1988). One of the more elusive elements of the design process is the inferred constraints that impact the design. Inferred constraints are critical to software design. These constraints are not given explicitly in the user requirements but are inferred from the domain or from the solution process (Guindon, 1990, p. 288). Capture of the inferred constraints that the problem domain imposes on the design process is one of the most important tasks of design rationale. These domain constraints must be explicit in any explanation of an information system in order to avoid the classic problem of "underestimating what the user doesn't know". The early identification of inferred constraints has proven to be one of the key tools used by systems design experts to narrow the solution space of the problem at an early stage (Guindon, 1990, p. 290).

Potts (1996) shows how the design rationale approach can be interleaved with various IS development methodologies. He shows how IBIS can be extended to account both for the stages of the IS development cycle, such as requirements, design, and construction, and for the evolution of the artefacts that emerge from these stages. Most importantly in the context of the research reported here, he describes an approach to relating IS software components to the design rationales that underlie their purpose and structure. This approach will be discussed further below and in later chapters.

Lee (1990) argues for the use of DR to support dependency management among the elements of a given design space. One of the more difficult problems that has been uncovered with the use of design rationale is with management of 'consistency links', representations of where certain design alternatives support or clash with design alternatives identified in other parts of the system (MacLean, et al., 1989). These links represent constraints and dependencies in the design alternatives under investigation. External consistency links describe how elements of the design space relate to the domain in which the system will operate.

Dependency relations are important for facilitating change to designs and system evolution (Gruber & Russell, 1996; Ramesh & Sengupta, 1995). One of the major issues that must be addressed by any IS specification approach is how to deal with changes to system requirements, changes to the design, and changes to the system itself. Many IS are used in domains that are constantly changing and evolving (Slagle, et al, 1990). IS artefacts including systems and documentation must be looked upon as living and rapidly evolving phenomena that require a strategic commitment of resources if they are to remain relevant (Paul, 1994; Spinuzzi & Zachry, 2000). The efficacy of DR in support of system customisation, maintenance and evolution is widely recognised (Buckingham Shum, et al., 1997; MacLean, et al, 1996; Fischer, et al., 1997). DR should capture both the logical structure, why things are the way they are, and the historical structure, why features replaced or modified other features over time (Lee & Lai, 1996).

Clancey (1983) highlights the difficulties encountered by knowledge engineers as they try to maintain and extend a knowledge base for which they do not possess the rationale behind its structure, strategy, and support knowledge. Systems require a "knowledge specification" to guide changes to a system in the maintenance phase of its lifecycle (Slagle, et al., 1990). Design is often characterised by "ah ha" events where a design team makes a breakthrough that reverses many previous design decisions (Conklin & Burgess-Yakemovic, 1996). Though DR provides explicit support for these types of discoveries, the DR itself must then be updated to account for these changed assumptions.

Design rationale techniques have been applied effectively to the problem of human-computer interaction (HCI) design (McKerlie, et al., 1993). Carey, et al. (1996) have assembled a library of HCI design problems and the rationale that led to their resolution. This library is meant both as an instructional tool for novice HCI designers and as an online reference for HCI designers engaged in development projects. The authors claim that their system providing access to generic HCI design rationales was effective in being able to provide help to users of the HCI design tool.

Simon (1996) claims that one of the best ways to develop design logics is through careful analysis of how designers behave they are working carefully (Schön would say "reflecting"). DR approaches have been used to reconstruct the rationale behind

an existing systems both to extract the design principles behind an artefact and to gain understanding to support redesign (Shum, et al., 1993). Applying QOC and Design Space Analysis can work as a form of reverse engineering when applied in retrospective studies (McKerlie, et al., 1993). Guindon (1990) suggests that the identification of a "primary position" e.g., reliability, performance, economics is one of the single most important factors affecting the design process. This task also helps by providing solution constraints early in the design process. McKerlie, et al. (1993) used DSA/QOC to perform a retrospective analysis of a set of educational hyper-systems, in this case all Apple HyperCard stacks. The aims of the project were to uncover the complexity in the design of such systems and to test the effectiveness of retrospective DR in design evaluation. The research team was able to identify six design dimensions in these types of systems: the purpose of the system, its content, structure, navigation, control, and presentation style. The study demonstrated the efficacy of DR approaches for extracting design principles from pre-existing systems.

Communicating

Wenger (1987, p. 7) defines knowledge communication systems as those that provide:

> *"...the ability to cause and/or support the acquisition of one's knowledge by someone else via a restricted set of communications operations."*

Shipman & McCall (1997) claim that DR's benefits are derived primarily from the documentation produced and its utility as a communications medium rather than from the extent to which it structures and improves the design process. Design rationale supports communication and therefore coordination between project team members, between team members and users, and between different project teams across projects, organisations, space, and time (MacLean & McKerlie, 1995; Ramesh & Sengupta, 1995). Graphical DR notations such as DSA/QOC help to make clear the most salient issues of a given design, thus making these easier to communicate both within and outside the design team (Buckingham Shum, et al., 1997).

Project teams using more standard IS modelling tools have identified the use of a common system representation as an important tool in the design process, especially for communication across project teams, though they were individually selective in their choice of a format for this information and acquiring the knowledge needed to use them often took considerable time (Curtis et al., 1988). Where DR formalisms are

being used to capture design deliberations, cases have been reported where designers became increasingly reliant on their DR representations as a form of expressing design issues (Conklin & Begeman, 1988; McKerlie & MacLean, 1993).

In applying DR (IBIS-based) over an extended period on a design project, Conklin & Burgess-Yakemovic (1996) found that DR both improved design meetings by providing an agenda and capturing results of previous meetings and assisted in the process of acclimating new team members to a project. Though some non-team users of the DR output complained that the format was "too technical", it was converted easily to prose and the results were effective in communicating the state of the issue.

One of the explicit goals of the IBIS and gIBIS approach is to help design members appreciate each other's views, focus on central issues (MacLean & McKerlie, 1995), and to minimise less constructive discussions, for example, name calling or axe grinding (Conklin & Begeman, 1988).

Lee (1997) argues that there are a host of potential users of design rationale products outside of the immediate design team. These include managers who can use the design rationale to evaluate both the process and the product of the design effort. Other potential user are lawyers, who can use design rationale in intellectual property and product liability cases. In many organisations, IS documentation often serves as dual purpose as both an instructional aid and as a marketing tool to help explain what features a system provides (Hackos, 1998).

Buckingham Shum et al. (1997) point out that given the 'matrixed' management approach employed in many organisations, where project team members, system users, and other stakeholders are continually shuffled between projects, keeping track of the most important issues affecting any one of these projects can become an intractable problem. Design rationale can serve as an organisational memory for a given design project. As DRs accumulate in a given organisation (or across organisations), DRs can be used to extract the principles that drive design in the organisation, and also the problems that inhibit effective design.

Fischer (1999) argues that DR and related approaches can support closer integration of IS producers and consumers and, ideally, evolve consumers (users and other stakeholders) into IS producers. This closer integration of IS producers and consumers

also means that the DR captured will be that much more complete and it includes the consumers' domain-oriented input to the design deliberations. The "Domain-Oriented Design Environments proposed by Fischer include a social model where producers and consumers of the IS product work together closely towards system evolution, creating, in essence, a micro version of the open source movement's community of practice. Examples of the use of DR to support closer integration of the stakeholders in a complex design activity include (Sjöberg & Timpka, 1995).

'Micro' approaches such as software component development and reuse, and 'macro' strategies such as corporate knowledge management share a common motivation and goal. Significant investment is made in the process of design, whether the artefact under design is a software system, a more tangible device or product, or an organisation itself. Design "in-the-large" involves coordinating the efforts of multiple designers and synchronising these efforts with the needs of the stakeholders in the problem domain. Knowledge is produced when these different actors come together to apply themselves to a problem, but typically only one facet of this knowledge, in the form of design *decisions*, is captured for later use. These decisions alone, no matter how well documented, may not be sufficient to answer *why* questions related to the resulting artefact.

As Curtis, et al. (1988) found, the critical knowledge needed to understand an IS relative to its application domain is often concentrated in the minds of one or two exceptional system architects. More general studies of organisational competence suggest a significant trend towards the control by expert knowledge workers of the critical information assets of the organisations that they serve (Albert and Bradley, 1997). In response to this trend, one of the most active areas of both research and practice in information systems today is corporate knowledge management. O'Leary (1998) cites the following reasons for the rapid growth of this field:

1. Environmental pressures – competition, downsizing and resultant loss of expertise, globalization and distribution,
2. Technological advances – Internet, intelligent agents
3. Value of information

The basic model of knowledge management and organisational memory involves providing tools (generally IS tools) and defining procedures to collect employees' knowledge in a central store, and then provide easy, rapid access to this store. One the key issues identified with this approach is motivating employees with different and potentially competing interests to contribute to the knowledge base (Orlikowski, 1996). Contributions to organisational memory need to show immediate and direct benefit to contributors for the approach to be effective (Grudin, 1994). However, software-based support for this process can help align the cost-benefit equation by making it easier to capture information and to retrieve it for later use (Fischer, 1999).

Research into complex IS development projects discussed earlier in this chapter suggests that a small number of project team members, one or two on any given project, are able to absorb and understand the complex array of domain factors that impact a system design, and how domain problems are addressed by the system's structure. One role for DR may be to explicitly support these individuals in gaining, capturing, and communicating the understanding shown to be so valuable. Some DR studies have reported on the importance of a "champion" to evangelise and support the use of DR approaches among project team members (Conklin & Burgess-Yakemovic, 1996). Perhaps these individuals are one and the same?

Design Rationale Usability

Bush (1945) noted that:

> *"a record, if it to be useful to science, must be continuously extended, it must be stored, and above all, it must be consulted."*

As Fischer, et al. (1996) point out, approaches to approaches to capturing argumentation-based design rationale require directed efforts on the part of the project team to make the products of DR more useful and relevant to design and construction. Very few extended studies of the effect of applying the DR approach have been completed. Conklin & Burgess-Yakemovic (1996) report that on one long-term project, several potentially costly design errors were identified through the use of DR and argue that these design errors would have cost 6 times that of using the DR capture tool. The project team also found the approach useful in reviewing requirements, in particular in helping to identifying the faulty assumptions and

missing rationale relative to the system requirements that drove the project (Conklin & Burgess-Yakemovic, 1996).

Knowledge communication relies on ensuring the fidelity of information as it is encoded into a representational mechanism and then subsequently decoded for use (Wenger, 1987, p. 286). Much of the research into DR has focused on the usability of particular formalisms and tools (e.g., Conklin & Begeman, 1988; Buckingham Shum, et al., 1997). Though as discussed above DR may help to structure and resolve "breakdowns" in the design process, the overhead of using a DR formalism and supporting software tools can itself lead to breakdowns, further complicating and disrupting the design process (Conklin & Begeman, 1988).

Clancey (1983) points out that by requiring domain experts, in this case system designers, to become 'taxonomists of their own knowledge', we ask them to assume a significant burden above and beyond that with which they are already tasked. He points out the difficulty in asking system designers to document even the very highest level strategies they employ, strategies that may be implicit in the work of even novice designers. Some DR studies have reported on the importance of a "champion" to evangelise and support the use of DR approaches among project team members (Conklin & Burgess-Yakemovic, 1996).

Gruber and Russell (1996) criticise DR approaches and formalisms that attempt to prescribe how the design process should be undertaken. They argue instead that what should be captured in DR is not information that fits a preconceived notion of the design process but the data and information that can be used to answer designers questions in whatever form they take. Olson, et al. (1996) found that design meetings are highly structured and consistent as is, and they question whether this makes it easier or more difficult to integrate DR approaches into the process. Shipman and McCall (1997) discuss the opportunity costs incurred by the DR approach, as the time spent recording DR could potentially be applied to other more relevant design activities.

Buckingham Shum, et al. (1997) report that the DSA/QOC is useful in attempting to work through the issues presented by ill-defined or difficult design questions, but that the approach become unwieldy when applied to simpler, well constrained issues. A major problem with the use of DR formalisms is building the structure of information

that rationalizes design decisions. Gruber and Russell (1996) cite studies that show that even motivated designers soon lose interest in capturing DR, they label this phenomenon "rationale fatigue".

Another key problem is the potential for detail of unmanageable proportions when analysing a design space (MacLean et al., 1989). They offer the notion of expandable detail as the solution to this problem. Users should be able to 'tunnel' deeper and deeper into a particular rationale to understand at increasing levels of detail why a design decision was made.

MacLean, et al. (1996, p. 55) argue that design space analysis pays for itself by supporting the design process, redesign, and design reuse. It does this by providing an explicit representation to aid with reasoning about the design and by serving as a communications medium for project stakeholders. However, there is a clear trade-off between effort expended during the design rationale capture phase and the design rationale use phase (Buckingham Shum, et al., 1997; Moran & Carroll, 1996). Moran and Carroll (1996) argue that a critical success factor is the extent to which design rationale tools can be integrated into day to day design activities:

> *"[the] general strategy for getting design rationale into practice is to embody rationale capture in tools that are of immediate utility to designers".*

Some approaches propose automated, AI-based methods for reasoning with DR. These approaches generally require more formal, or mathematical representations of the DR that typically must be retrospectively derived from more information representation (Ramesh & Sengupta, 1995). Despite the obvious incremental cost of capturing DR to the level of detail required for these systems, more formal DR representations may have a bigger payoff, despite their greater cost, as AI-based DR systems can reason deductively about the validity of a design (Ramesh & Sengupta, 1995).

Karsenty (1996) studied the use of design rationale (QOC) on a small redesign project in mechanical engineering. The study investigated whether engineers *need* the design rationale to understand a device, whether they *use* available DR documents in the process of gaining understanding, and whether the rationales captured are appropriate to the understanding task. The study found that during engineering meetings, over half of the questions asked by engineers could be classified as design rationale questions

and approximately 40% of these questions, presumably unanswerable by other available engineering documents, were answered by the available DR. Problems they found with the DR related to both how it was captured and used, and were largely based on an incomplete capture of the issues important to the design. They found that engineers used DR in two ways, which they classify as opportunistic and extensive. Opportunistic use is when engineers went to the DR because other available materials, e.g., blueprints, did not answer their questions. Extensive use occurred when engineers used the DR to gain a more in-depth understanding of the design that could be achieved with other materials. Overall, the found that DR was most often used in conjunction with other design materials. Especially on this redesign task, they found the DR helpful in explaining why certain features were being changed relative to the original design.

In contrast to Karsenty's results, Herbsleb & Kuwana (1993) found that DR-related questions occurred infrequently relative to questions related to software requirements, use scenarios, and system functionality. Their characterisation of DR questions was as those that corresponding to why questions. As Draper (1988) points out, most explanation requests can be formulated as why questions. Their analysis was based on data collected as project meeting (38 meetings) minutes with a smaller proportion from videotaped meetings (3 meetings). As pointed out by the authors, meeting minutes potentially do not represent a good source of DR data since they are often incomplete and based on the perception of the minute taker. However, they found significant agreement between their two data sets. The authors point out that a possible explanation of the low number of why questions is that these correspond to implicit or tacit knowledge that designers bring to the task but do not make explicit in their design deliberations.

In a study of the development of the International Classification of Disease (ICD) system by the World Health Organization (WHO), Bowker and Leigh-Star (1994) explore one of the basic problems with coding formalisms such as those employed by DR. The ICD is distributed to health care organisations around the world and consists of a numeric scheme for classifying disease as well as algorithms for determining cause of death in cases where multiple diseases occur. The WHO experienced significant problems with standardizing local use of the standardised coding scheme because of local practical issues, cultural norms, and even taboos that affected how

the coding frame was applied. For example, sexually transmitted diseases were often underrepresented in coding results. A less obvious example was the reluctance in Japan of coding cause of death due to heart disease as the local connotation is that the victim lived a life of physical labour. Bowker and Leigh-Star argue that standardized coding mechanisms must allow for a degree of customisation. In effect, ambiguity must be designed into the coding frame if it is to be successful. The authors argue that rather than trying to eliminate ambiguity, the goals should be to define the level of acceptable ambiguity in order to meet local requirements.

Fischer, et al. (1996) found that in one study, participants seemed to reject the DR approach (in the case PHI), especially as project development become more construction oriented, however, in analysing the protocols from the study, these participants seemed to be applying PHI as they had been taught. They explain this using Schön's theory of "knowing-in-action" (Schön, 1983) and argue that the results of their suggest that at least a limited extent, the reflective approach which is a foundation of DR had been internalised by the design team members.

Though DR approaches represent an additional overhead in the design and development process, its originators argues that DSA pays for itself by supporting the design process, redesign, and design reuse by providing an explicit representation to aid with reasoning about the design and by serving as a communications medium for actors involved in the development process (MacLean, et.al, 1996). However, more research is needed into the how well DR approaches can become integrated into the design process of a community of practice, and what pay-offs accrue when the approach is applied over extended periods.

Detailed reviews of the usability of design rationale formalisms can be found in (Buckingham Shum, 1996; Buckingham Shum & Hammond, 1994 ; Shum, 1991).

4.3 Explanation and Design Rationale

Whereas much of the DR research focuses on the use of design rationale for process improvement within design teams and for communication between designers (e.g., MacLean 1989; Guindon, 1990), this thesis explores the use of design rationale to support computer-generated explanations of an information system. Though the potential contributions of design rationale to communications between designers of a

system and system users and other external stakeholders has been previously identified (e.g., Moran & Carroll, 1996; MacLean & McKerlie, 1995), and explanation has been identified as a key by-product of the DR approach (Mostow, 1985; Gruber & Russell, 1996), little research has explored the form that these explanations might take. This section explores this potential, in particular the extent to which design rationale may capture the explanatory information identified in chapters 2 and 3.

Several IEF research projects discussed in Chapter 2 identified the activities of the design phase and design knowledge capture as the key to producing system explanations, in particular (Swartout, 1983). Chandrasekaran and Swartout (1991, p. 47) argue that:

> *"Knowledge systems based on explicit representations of knowledge and method, with information about how and from what their knowledge was obtained, are the foundation for producing good explanations."*

One of the elements in the Explainable Expert Systems project (Swartout and Smoliar, 1987) is the use of an automatic program writer that maintains a log of the design decision made during the expert system development process. This tool attempts to capture the knowledge lost during expert system development in a high-level representation from which explanations may later be constructed. This experimental tool was based on the proposition that the knowledge needed for an IEF becomes known to the system builders during development, but because it is not strictly necessary for the system to perform its specified task, it is not integrated into the system. This lost knowledge includes principles and facts of the problem domain and, most importantly, how these principles and facts are translated into a working system. This section explores how this same information might be captured manually using the design rationale approach.

Gruber (1991) argues that capturing design rationale is a knowledge acquisition problem and is explicit as to the contribution design rationale makes to the explainabilty of a system:

> *"I frame the problem of capturing design rationale as a knowledge acquisition problem, where the task is to elicit, from domain specialists (designers), knowledge that enables a program to generate explanations of how the designed artifact is intended to achieve its function" (Gruber, 1991, p. 70).*

To Gruber, system explanations should describe how the system's components interact and how the system's behaviour contributes to a solution in the problem domain. According to Zimmer (1989), IEF explanations fail when they do not include sufficient representation of the system's underlying principles, when they make erroneous assumptions about the user's world knowledge, and when they neglect of principles of communication.

In a retrospective on his *Epistemology* paper, Clancey (1993a) refers to support knowledge as the *design rationale* that justifies the existence and representation of a given rule. Support knowledge grounds a rule, and therefore its explanation, in both physical world facts and the social context in which it is used. A common theme in the IEF and ITS literature is that a lack of system transparency, and therefore explainability, arises due to the disjunction between the strategy of a system (problem solving approach in KBS, the pedagogical approach in ITS, and the domain or business rules in IS) and the system's structure, how it is engineered to implement the strategy (Clancey, 1983; Wenger, 1987). Clancey later described this missing link as the *focusing principles* needed to bridge "what is true", the domain facts, and "what to do", the inference procedure operating on those facts (Clancey, 1986, p. 58). The ability to explicate this implicit knowledge provides the system developers and domain experts with a better understanding of the problem, a more directed and higher performance reasoning process, and more user-oriented explanation systems; they also add significantly to the clarity and thus the maintainability and extendibility of the finished system. During design, DR helps to capture this deep system structure and control knowledge as it is brought to the surface during design deliberations.

The motivations behind providing design rationale explanations to those external to a project team, such as end users, are derived in large part from the socio-technical approach to IS development (e.g., Lin & Cornford, 2000; Avgerou & Cornford; 1998) and participatory design (e.g., Mumford, 1996; Greenbaum & Kyng, 1991). The former considers humans as an essential, and essentially *human*, part of an overall system. The latter promotes a more inclusive system and work design process both to produce better systems, and to promote ethical and democratic organisational dynamics. While exposing the rationale behind a particular design may not result in a system that is immediately easier to use or more useful, the inclusiveness fostered by this openness may play an important role in bringing system developers and users

together into more informed "communities of practice" (Lave & Wenger, 1991) that can positively impact the evolution of a system and the form that future systems take. These ideas will be explored further in the chapters that follow.

The following diagram shows how design rationale may be situated in the context of the IS development process and how the explanation content model developed in the last chapter relate to both DR and to the development context.

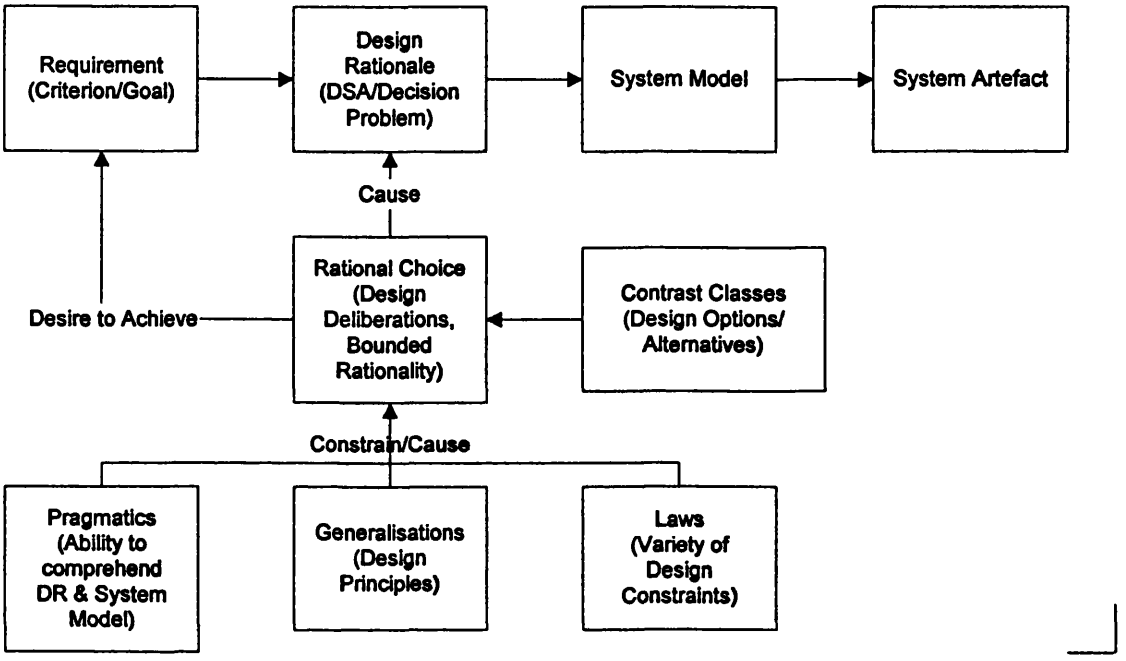


Figure 11 - Design Rationale & Explanation Theory

4.3.1 Design Rationale as Causal Explanations

Jagodzinski and Holmes (1989) argue that within the field of expert system development, systems are often based on shallow knowledge that does not include the underlying causal model of the domain. Without access to these causal models, an expert system must resort to *ad hoc* data in attempting to explain its conclusions. Gruber and Russell (1996, p. 330) argue that:

> *"Rationales are not just statements of fact, but explanations about dependencies among facts."*

Causal interactions between the components of an IS are an important element of system explanation (Gruber, 1991). Clancey (1983) found that in order to provide

explanations, the features of a problem domain, the relationships between these features, and the nature of these relationships relative to the system structure needs to be made explicit. DR has been used to understand the causal links that exist between components in a complex architecture and to support human reasoning about the effects on the architecture when new components are added (Bose, 1998). Though Lee (1990) argues that designers find *explicit* consideration of causal chains unnatural in the design process, links between the form of a mediating artefact and the problems the artefact is meant to address is an essential component of useful system documentation (Draper, 1998; van der Meij & Carroll, 1998).

Design rationale provides several important forms of causal information. First, by relating the form that system takes to the way in which requirements were interpreted during design deliberations. While these interpretations may not be correct in the strictest sense, access to DR does open the "black box" of requirements interpretation for those who seek to understand this transformation. Second, by its nature DR makes apparent the design deliberations that caused a particular system model to emerge from the variety of constraints that impact these deliberations. Finally, DR exposes the causes of why certain forms *were not* adopted in the design.

4.3.2 Design Rationale as Deductive-Nomological Explanation

In Chapter 3 we discussed how the received view of scientific explanation, the deductive-nomological (D-N) account makes use of laws, antecedent conditions, and deduction to construct a an explanation. We also discussed how in the social science, the criteria for what qualifies as a law is relaxed to include well-established universals. Huberman and Miles (1994) take the view that there are law-like, stable relationships that exist in the domain of social phenomena. They argue that while social knowledge must be considered in historical context, there are identifiable processes that transcend any singular context and are applied from one context to the next. In the IS context laws can take the form of standards (technical or professional), methodologies, social norms, or even existing designs that act as exemplars. Physical as well as social laws play an important role in the design and development of an IS in that they act as constraints or selection criteria on the design alternatives that are considered.

Engineered devices are subject to the same laws, in the form of constraints, as the rest of the physical world (Simon, 1996). Of the more elusive elements of the design process are the inferred constraints that impact the design. These constraints may not be explicitly mentioned in the requirements but are inferred from the domain or from the solution process (Guindon, 1990). Capture of the inferred constraints that the problem domain imposes on the design process is one of the most important tasks of design rationale. These domain constraints must be explicit in any explanation of an information system in order to avoid the classic problem that Guindon calls "underestimating what the user doesn't know". The early identification of inferred constraints has proven to be one of the key tools used by systems design experts to narrow the solution space of the problem at an early stage (Guindon, 1990, p. 290).

Gruber and Russell (1996) present an model for generating deductive explanations conforming to Hempel's D-N model (see Chapter 3) that relies upon formal engineering specifications of the system being explained. Though the model is only cost-justified in cases where such formal models already exist because of the expense or safety-critical nature of the device, their view of device explanations provides some insights into the nature of a system explanation. In particular, they see explanations as the interaction or *deduction* of the functional role of a system or component and the limiting constraints that bound how the component may behave relative to the domain requirement it addresses .

Conceived of in this way, laws relate to the design rationale, or argument structure, which can provide a robust description of the reasons why a system operates as it does. In a given system design space structured according to DR formalism, laws may take the form of what Toulmin called a *backing* (Toulmin, 1958). In DSA/QOC (MacLean, et al., 1996) an assessment of a design option relative to some criteria may be justified with an argument. These arguments can take the form of well-established theories or empirical studies (laws) that support, and explain, how the choice of a particular option was established. One example of DR in use shows how psychological theories directly impact the form of user-system interface components (Carroll & Rosson, 1996). Buckingham Shum et al. (1997) report that in design exercises using the QOC DR formalism, team members engaged in a constant process of evaluating design options relative to constraints. Shum et al. (1993) found that certain *principles* are foundational issues related to the application domain that guide

the design. The design team constructing the retrospective DR felt that access to these principles and themes as applied in prior design rationales would be very useful for new application projects or enhancements to the existing process.

Only by understanding the justification for a law can an expert know when they can break it (Clancey, 1983) and what the consequences of breaking it are (Clancey, 1986). Design rationale as explanatory content has the potential to show where a system is being asked to perform beyond the limits of its design. This feature was shown to be particularly important in domains with a safety-critical element, such as medicine (Swartout, 1983). By explaining with laws, design rationale explanations help to justify design decisions by showing what was considered possible within the constraints or laws imposed by the physical, technical, organisational, and social environment.

Are Design Generalisations Laws?

In the IEF literature, several researchers contend that in explaining the operation of a KBS, it is useful, and may even be necessary, to develop generalised abstractions of the underlying process model and make these explicit in the explanation knowledge base (Clancey, 1983; Swartout, et al., 1991; Swartout & Smoliar, 1987). One of the mechanisms shown to be useful in a system explanation is to highlight the relationship between abstract and concrete knowledge and how they are applied to system behaviour (Lamberti & Wallace, 1990). Clancey (1983) argues that, counter to our intuitions, explanations that refer to general phenomena may actually provide additional detail by referencing more fundamental phenomena of which we may already be aware. Of course it is possible that in the IS development context one class of stakeholders, for example the system's developers, may exhibit an entirely different understanding of how a particular entity, process, or event relates to given generalisation. For example, while developers may see a particular design decision as an example of sound security practices, users of the same system might relate it to the generalisation "the IS department always makes it difficult for us to access our own data." A recurring theme in expert systems explanation research is how to relate the domain-independent explanatory principles (such as general problem solving strategies) represented in the IEF to the domain or application-specific information

that shows how a general principle was applied to solve a particular problem (Clancey, 1983; Swartout, 1991).

Design rationale makes explicit how general principles affected the form of a design. For example, DSA/QOC (MacLean, et al., 1996) includes the concept of bridging criteria that relate general design goals to a specific design question and Fischer, et al. (1996) describes the JANUS system, which uses design critics to produce explanations based on comparing an evolving DR to predefined constraints corresponding to good design practice. MacLean & McKerlie (1995) show how DSA/QOC may be used along with other design artefacts to develop generalisations to guide the design process. Potts (1996) specifically relates design rationale to the abstractions that form the basis of software systems. These generalisations are both developed and captured through the use of design rationale techniques.

Shum, et al. (1993) found that designers were constantly guided by certain *themes* such as flexibility and maintainability. Carroll (1998) describes the concept "usability specifications", for example: 'users can create invoices with the system with an error rate of less than one percent', that can be operationalised first at the meta level and then in correspondence with a particular system window, or even an individual field or button. For example, the specification above might be operationalised as 'the Save button will be disabled until the required fields on the invoice field have been completed.' Such statements are explanatory in that they provide the user with the motivation, the 'why', behind what could turn out to be an unpopular feature (e.g., if users' job performance is measured based on the number of invoices captured per day). Access to the generalisations or meta-criteria that guide design may help users to construct their own understanding of how a system fits into a given problem domain (Carroll, 1998). Software online help frequently act as a bridge between the complexity inherent in sophisticated information systems and the simpler, more general concepts more comfortable to many people (Patrick & McGurgan, 1995).

4.3.3 Design Rationale as Pragmatic Explanations

As discussed in last chapter, explanations conforming to the pragmatic theory are those that are most closely tailored to the requestor's need and context. Other elements of the pragmatic approach is the use of relevance relations between information elements and contrast classes to help vector an explanation relative to

how a given fact could be otherwise. Clancey (1983) argued that when providing a causal explanation of MYCIN's processing strategy, the system should include the ability expand a rule indefinitely, providing increasing levels of detail as requested by the user. He argued for example that experts work very differently from naïve problem solvers, in particular, they are typically unaware of their use of first principles in attempting to work through the problem and the information they seek tends to be specific to the problem at hand.

Design rationale capture and delivery techniques and tools typically make use of the node-link structure known as hypertext (Shum, 1991). One of the advantages of passive information delivery mechanisms such as hypertext is that users are to a great extent able to direct their own information seeking behaviour. Several studies have identified hypertext as a method for delivering explanations that allow for self-selected information use both for IEFs (Moore, 1995) and for systems generally (Grice, 1989).

DR and Explanatory Relevance

Some researchers into IEF explanations claim that explanations of knowledge-based systems can be improved by relating system behaviour to facts in the problem domain (Lamberti & Wallace, 1990). Some work has explicitly explored the way in which a DR formalism can be used to relate use scenarios or task descriptions to the DR argument by representing task requirements with design criteria (MacLean & McKerlie, 1995; MacLean, et al., 1996; Carroll & Rosson, 1996). These approaches attempt to establish a relevance relation between the explanatory element, for example, the questions, options and criteria, and a potential explanation request motivator, typically the domain task.

In one case of reconstructive DR using QOC (Shum, et al., 1993), project documents and other supporting materials were analysed and structured into a DR using the following techniques:

1. Filtering – reviewing the relevance of different sources
2. Integrating – organising sources by concept
3. Indexing – building links into the body of the source document

Techniques such as this can help to structure and focus what can become an extremely large and complex knowledge base related to a design. Minsky (1961, cited in Stefik & Conway, 1982) argues that the key to solving the combinatorial search problem inherent in complex information domains is the identification of what he calls *"planning islands."* The concept of planning islands in Minsky's AI context refers to heuristics embedded in a knowledge base that help to prevent dead-end searches. Unfortunately, as Clancey (1983) pointed out and we discussed in Chapter 2, these heuristics often take the form of meta-rules or rule order structure which make implicit the justification for the heuristic.

This problem has been addressed in a number of ways in design rationale and design space search. In QOC, criteria and bridging criteria can be used to explicitly represent the relationship between a system requirement, the design questions that emerge from it, and the deliberations behind their resolution (MacLean, et al., 1996). Carroll and Rosson (1996) show how design rationale can be linked to domain tasks and the elements of the user-system interface that support performing the task. Another important example comes from Potts (1996) who shows how design rationale can be linked directly to the software objects, including classes, procedures, data structures, and other abstractions, that include, for example, the steps of a methodology applied in the design process, or the statement of user requirements that guides design of the artefact. In attempting to identify design rationale fragments relevant to a given feature, the user requirement realised by that feature can help to guide the search. In general, planning islands represent evocative inscriptions embedded in the knowledge base that help to guide and lessen the cognitive load of its users.

DR and Contrast Classes

The concept of capturing contrast classes is central to design rationale approach and is embedded in the techniques and tools that have developed to support the approach. For example, IBIS's *positions*, QOC's, *options*, and DRL's *alternatives* each provide a placeholder for information relating to "how it might otherwise be" (MacLean, et al., 1996, p. 55). van Fraassen (1991) argues that contrast classes provide the "identity conditions" for the event to be explained. Contrast classes provide information on *why* a particular event occurred instead of, or in relation to, another in its contrast class (Gasper, 1991).

In the IS context, the things that a system *cannot* do was found, unexpectedly, to be an important component of help system content (Roesler & McLellan, 1995). Lamberti and Wallace (1990) also highlight the utility of presenting alternatives in the context of KBS reasoning decisions, and why they were not chosen in a given problem solving session.

4.3.4 DR as Functional Explanations

Kroes' approach to technological explanation echoes that of Dennett (1987) and as extended by McCarthy (1991). This approach describes the stances that one takes towards an object or system. Dennett's original three stances are:

1. Physical stance – the physical or structural nature of the object is considered
2. Intentional stance – the object is considered in virtue of its beliefs, goals, and intentions
3. Design stance – what components make up the object

To this framework McCarthy added the following:

4. Functional stance – what does the object do

Kroes (1998), following Vincenti (1990), argues that there is a missing link, what he calls the *operational principles*, between the structure of an artefact and its function that prevents either being derived from the other in any logically valid process. These operational principles exist outside of the set of purely physical and scientific knowledge that make up the structural component of the artefact, they are represented in the technological requirements that drive the design. Since form does not follow function and function does not follow form, a D-N type explanation of an artefact in either of these directions is impossible (Kroes, 1998). Likewise a purely functional explanation is unable to provide the necessary linkage between the structure of a system, its function, and its operational principles.

Functional explanations provide the motivation behind a particular system feature or process and help to relate the feature or process to the tasks, usage scenarios, or business processes envisaged as part of the application domain. Artefact designers

sometimes become so immersed in the form and structure of their designs that they lose track of the functional motivations for behind their efforts, the original problem they set out to address (Patrick & McGurgan, 1993). Gruber (1991) argues that explanations should show how a system's behaviour contributes to a solution in the problem domain. Design rationale has been used to directly relate system design decisions to usage scenarios in the system domain (MacLean & McKerlie, 1995).

Lee's (1997) multi-layer model of the design space include the "design intent layer" where design alternatives are evaluated relative to project objectives. According to Bradshaw and Young (1991), a device representation strategy draws on knowledge of purpose and knowledge of structure to describe the device in question. Knowledge of purpose in the case of an information system design should result from the system's requirements definition phase. According to Gruber and Russell (1996, p. 337):

> *"One of the primary uses of design rationale information is to communicate the intended purpose or expected behaviour of the design."*

Knowledge of structure involves understanding a system's components and how they interact to achieve the system's stated purpose. Potts (1996) shows how design rationale can be linked to the structure of software components in an IS as well as how the development process affected their evolution. Earlier in this chapter we discussed how design rationale can be used to capture the laws and generalisations that guide the design process, these may form a significant component of the operational principles that Kroes argues are missing from a technological explanation that makes use of only information related to structure and purpose.

4.3.5 DR as Rational Choice Explanations

Identification of the factors that must be synthesized in order to produce an explanation of an IS is one of the central goals of this thesis. As discussed so far in this chapter, these factors may take several different forms and each may impact the structural, functional, and operational aspects of the system. During design, the process by which these factors are assigned relative weights is one of the components of an explanation of that artefact. Rational choice theories of explanation claim that people assign discrete values to particular outcomes, assign plausibility or truth values to propositions on a ratio scale, and choose among competing alternatives based on the highest expected utility of alternative (Davidson, 1974). However, Davidson

acknowledges that human intentionality and the fact that the beliefs and desires that drive the decision process change over time confound any attempt to predict or explanation the process that they will follow in assigning these weights.

By capturing the detail of design deliberations, DR information exposes the way in which a design team identifies intentions relative to the functionality required by the system, then considers how various reasons, or criteria relate to their ability to achieve these intentions. While the DR approach does not possess a "silver bullet" to make the actual intentions (or beliefs, or desires) of a design team explicit, the process of representation and reflection upon what the team *claims* are their intentions does help to expose at least that which they are willing to make public. Information of designers' intentions, such as that provided by design rationale, helps to form beliefs that map more closely to the original system model and therefore may assist users in taking more effective actions (March & Smith, 1995).

Tazi and Novick (1998) provide a brief account of a project that employed DR coupled with speech act theory (Searle, 1969) in an attempt to explicitly capture the communicative intentions of a complex documentation set for an aircraft flight crew. The supporting rationale captures the "intentional structure" of the documentation in terms of its format, structure, and expressive style in relation to the effects the designers which to have on their readers. They describe these intentions as "domain acts", which might include INFORM, DO, CHOOSE MODE, WARN, or VERIFY. The idea of making explicit not only the design rationale that is being employed but the potential communicative intention behind capturing the rationale provides a significant communicative operator available when a mechanism for delivering this information is implemented.

4.4 Discussion

According to Strauss and Corbin (1994) a theory "consists of *plausible* relationships proposed among *concepts* and *sets of concepts.*" This section is an attempt to synthesise the ideas introduced in the previous chapters into a coherent model of how explanations might be defined, structured, and delivered in an information system, and their content captured using concepts, techniques, and tools derived from design rationale. The theory developed in this chapter is an integration of ideas from the

study of IEFs, formal theories of explanation from philosophy, and design rationale, which have been packaged into an account of how self-explanations might be produced by a software information system. Prior research into IEFs provides the link to the most directly applicable research foundations for the work reported here. Philosophical theories of explanation provide rigorous structure, goals, and metrics upon which the account of explanation developed here is based. Design rationale, it is argued, provides the content that it is captured, represented, stored, and accessed to populate the structure of the explanation model.

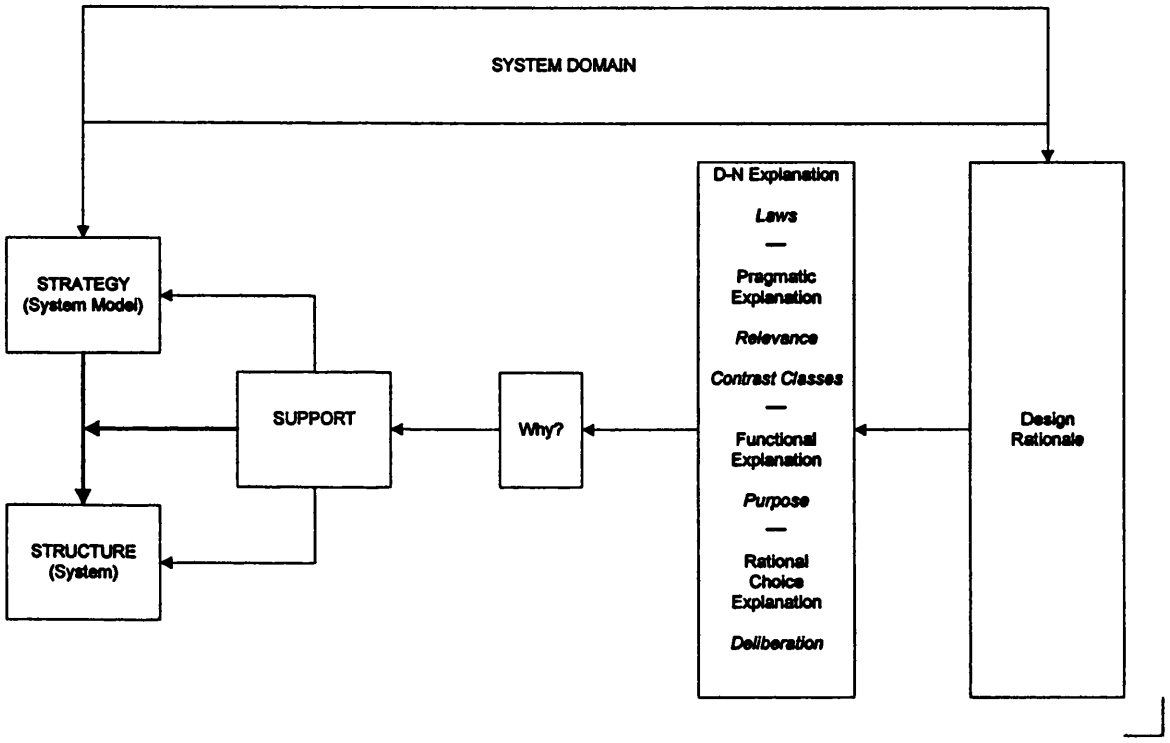


Figure 12 - A Framework for IS Explanations based on Design Rationale

In Chapter 2 we saw how Clancey (1983) conceived of expert system explanation deficiencies as stemming from the lack of support knowledge to describe how a system strategy is transformed into an actual system structure. Swartout (1983) saw the problem as a lack of *justification* available to answer the *why* questions related to the nature of this transformation. Chapter 3 explored theories of explanation that attempt to construct an idealised answer to a why question, and related to theories to the IS development and use contexts. This chapter began to explore the theoretical and empirical grounding of design rationale in an attempt to relate this approach to the explanatory information requirements suggested in Chapter 3.

The major premise of this thesis is that the process and products of design rationale are a potential source of explanatory content for a software information system. Design rationale models are argument structures, MacLean, et.al (1989) calls design rationale "the argument behind the artifact". Hempel believed that explanations are a form of argument (Ruben, 1993). Gregor and Benbasat, (1999) contend, based on prior IEF studies, that KBS explanations conforming to Toulmin's model are considered better justifications of the explanatory account, and that systems that provide these types of explanations may enjoy higher levels of user acceptance and user satisfaction. The proposition explored in the next two chapters is that the information needed to provide explanations of an information system can best be captured during the analysis, design, and construction phases of the system development process.

Design rationale as considered in this thesis is taken to include not only the rationale behind the decisions made in constructing a program, but all of those modes in the IS development lifecycle. These include but are not limited to: the original motivations for development of an IS, sometimes referred to as the problem statement that justifies the project in the first place; the rationale behind how requirements for a new IS are collected and the criteria that are used to select which requirements will be realised in the finished system; the technical, psychological, group, organisational, social, political, and other factors that constrain the process of design in the 'real world' context. Design rationale may contribute to the explainability of a system by the production of an annotated record of the design process that is an explicit representation of the explanatory factors and decisions that affected the design. The two chapters that follow will investigate these claims empirically.

5 Study One – A Closer Look at IS Explanations

This chapter describes the first empirical component of the thesis. The purpose of the study described in this chapter is to expand, refine, and evaluate the theoretical framework developed in the last three chapters using data collected from semi-structured interviews with IS development professionals. Recall from Chapter 1 the three research questions motivating the thesis:

> *Q1. What can philosophical theories of explanation contribute to the development of a framework for integrated explanation facilities?*
>
> *Q2. Can ideas from the fields of explanation systems and design rationale be integrated into the framework from Q1 to produce an implementable model for explanations of information systems?*
>
> *Q3. Is the model from Q2 operationally realistic, is it cost-effective, and can it be integrated into the IS development process?*

The purpose of the study described in this chapter relative to Q1 is to explore how IS professionals define explanation, and the issues that arise in capturing and delivering explanatory content. Relative to Q2, the goal is to investigate the potential role of information from the analysis and design phases of an IS project, in the form of design rationale, as explanatory content. Finally, relative to Q3, the study attempts to explicate the factors that study participants believed would support or inhibit the capture and delivery of explanations based on design rationale. In this chapter and throughout the remainder of the dissertation, the theory being developed, that of providing IS explanations based on design rationale, will be referred to by the acronym *DREX*.

This study involved a series of semi-structured, in-depth interviews with twenty seven IS development professionals. The interview guide used in the study was designed to explore the different facets of IS explanation as described in the previous chapters. The data collected in the study were analysed using qualitative techniques derived from the grounded theory approach (Glaser & Strauss, 1967; Strauss & Corbin, 1998) and further refinements of these techniques developed by Miles and Huberman (1994). This chapter first gives an overview of the grounded theory approach, then provides details of how the techniques and tools were applied in this study. These

sections are followed by an analysis of the results obtained from the study, and the final section is a discussion of these results relative to the theoretical framework developed in previous chapters.

5.1 The Grounded Theory Approach

Proponents of qualitative methods often argue that these methods allow the researcher to explore their subject "in context" (Hoepfl, 1997; Frechtling & Sharp, 1997). Field studies of how systems are developed and used, and ones that consider their situated, socio-technical nature, are essential to understanding how explanation facilities can be made more effective (Clancey, 1993b). Qualitative field studies of the IS development and implementation process are well represented in the literature (e.g., Curtis, et al., 1988; Wixon & Ramey, 1996), as are those which employ the grounded theory methodology (e.g., Orlikowski, 1993). Since the focus of information systems research has evolved from purely technical topics to those related to psychological, organisational, social, and managerial issues, the use of qualitative techniques has increased (Myers, 1997; Galliers & Land, 1987). Qualitative research methods are used both in information systems research, and in the requirements, design, and usability testing phases of actual information systems development projects (Myers, 1997; Wixon, 1995).

A central goal of this thesis is the development of a theoretical framework to describe the content of IS explanations and to relate this content to the products of the design rationale approach. Remenyi and Williams (1996) argue for a stepwise approach to theory development such as the one reported here. In their continuum, research into new or relatively undeveloped topics proceeds as follows:

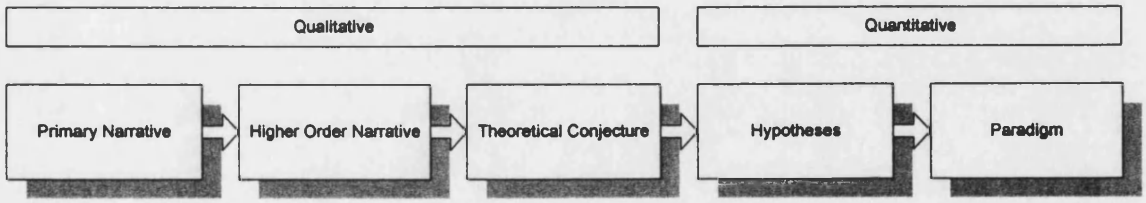


Figure 13 - Qualitative to Quantitative Research Continuum

They argue that creativity in science is a result of narrative thinking that supports ideas as they emerge and develop. Narrative accounts are condensed first into a higher order narrative and finally into a theoretical conjecture. Theoretical conjectures are

then used as the basis for hypotheses that can be tested using more traditional, quantitative research methods. The result of rigorously tested hypotheses is an established scientific paradigm.

Grounded theory (Glaser & Strauss, 1967; Strauss & Corbin, 1998) is an approach to qualitative research that uses field data as the building blocks for robust theoretical postulates. Strauss and Corbin (1998, p.12) argue that:

> *"Theory derived from data is more likely to resemble the "reality" than is theory derived by putting together a series of concepts based on experience or solely through speculation (how one thinks things ought to work)."*

Grounded theory is centred on an iterative cycle of data collection and analysis referred to as the *constant comparative* method. The approach relies on *analytic induction* (versus deductive or verificationist approaches) to explicate meaningful findings from data collected in the field (Berg, 1998). Although the approach is most often used to generate theory in relatively undeveloped fields or topic areas (Remenyi & Williams, 1996), it can also be used to explore and elaborate upon existing theoretical frameworks. Grounded theory involves development of hypotheses and attempts at their verification iteratively throughout the study, in contrast to the separate stages of hypothesis generation and test that form the basis of traditional scientific method. The approach is seen to fill the gap between purely speculative, and purely deductive research (Strauss & Corbin, 1998).

5.1.1 Theory and Code Frameworks

The use of grounded theory to inform or confirm existing theoretical frameworks is controversial. In its original form, the grounded theory approach prescribed against the development of any preconceived conceptual framework before field data have been collected and analysed (Strauss & Corbin, 1998; Glaser & Strauss, 1967). Others concur, arguing that when conducting and analysing the results of semi-structured, qualitative interviews, researchers should not enter the process with any form of *a priori* conceptual framework (Fontana & Frey, 1994). In particular, Strauss and Corbin (1998, p.22) caution against being led too strongly by existing meta-theoretical frameworks (e.g., critical theory, feminism). Huberman and Miles (1994) cite examples of qualitative analysis that have been performed in similar domains but from the perspective of researchers immersed in the ideology of competing meta-

theoretical frameworks. Despite similarities in the design of these studies, results differed widely in their conclusions and these differences are plainly identified as owing to the theoretical departure points of the different researchers.

Another perspective suggests that a myriad of factors already act as a lens through which the researcher interprets participant responses (Miles & Huberman, 1994), and that it is better to make the analysis framework explicit than to allow an implicit (to the reader), possibly theory-driven framework to impact presentation of the data and results (Lofland, 1971, Stratton, 1997). These factors include researchers' exposure to the literature of the field, conversations with colleagues, observation of practitioners, even the tape transcription process. Marshall and Rossman (1995) argue that in a grounded theory approach, the researcher may apply the results of the literature review to the development of initial concepts that can be used to 'prime' the study before empirical work progresses. Though Strauss and Corbin (1994) cautioned against these pre-empirical frameworks, they acknowledge that "theoretical sensitivity" is in fact an effective tool that can be applied by the researcher well versed in both the discipline and practice of their field of study.

Miles and Huberman (1994) suggest that to avoid qualitative data overload, a common problem especially for novice researchers, development of an initial conceptual framework can help to focus data collection, analysis, and management. This type of qualitative analysis is sometimes known as *theory-led thematic analysis* (Hayes, 1997). The approach requires, however, that the researcher be diligent in considering the effects of "tunnel vision, bias, and self-delusion" (Miles & Huberman, 1994, p. 56). The study reported her employed a theoretical framework as described in the preceding chapters. Details of how this framework was incorporated into the method are reported later in the chapter.

5.1.2 Data Collection

Semi-structured, in-depth interviews allow the researcher the opportunity of follow-up questions when responses to pre-defined questions illuminate a topic for further review (Mahoney, 1997). In qualitative research generally, as in grounded theory, the use of more open-ended questionnaires to research questions in fields without a strong base of theoretical work is widely supported in the literature (Marshall & Rossman, 1995; Oppenheim, 1992; Sudman & Bradburn, 1982; Strauss & Corbin, 1998). One

argument for the essentially different natures of exploratory and confirmatory research is from Oppenheim (1992, p.67), he claims that:

> *"The purpose of the* exploratory *interview is essentially heuristic: to develop ideas and research hypotheses rather than to gather facts and statistics. It is concerned with trying to understand how ordinary people think and feel about the topics of concern to the research."*

Questionnaire Design

Opinions differ on the degree of structure to be imposed on qualitative instrumentation such as questionnaires, or well-defined interview guides. Miles and Huberman (1994) provide a concise table of decision factors that they believe should drive decisions about instrument structure. A ♦ symbol next to cell in the table above designates where the criterion applies to this study.

	Less Structured Instrumentation		**More Structured Instrumentation**
♦	Rich context description needed		Context less crucial
♦	Concepts inductively grounded in local meanings	♦	Concepts defined ahead by researcher
♦	Exploratory, inductive		Confirmatory, theory-driven
	Descriptive intent	♦	Explanatory intent
♦	"Basic" research emphasis	♦	Applied, evaluation or policy emphasis
	Single case	♦	Multiple cases
	Comparability not important	♦	Comparability important
	Simple, manageable, single-level case	♦	Complex, multilevel, overloading case
	Generalising not a concern	♦	Generalisability/representativeness important

♦	Need to avoid researcher impact		Researcher impact of less concern
♦	Qualitative only, free-standing study		Multimethod study, quantitative included

Table 2 - Criteria for Qualitative Instrument Selection

As suggested in the table above, application of these principles is not straightforward, many of the criteria cross over, and many are arguably applicable or desirable traits of any study, regardless of its essential nature.

Sampling

No clear guidelines exist for the determination of a sample sizes and or sampling frames for qualitative research, and this lack of consistent guidelines has been identified as one of the key problems with qualitative approaches (Miles & Huberman, 1994). Purposive sampling is one accepted approach in qualitative studies, it involves the researcher applying their special knowledge of a field to select participants that best represent the different segments of the target population (Berg, 1998). These are also known as *judgement samples* (Oppenheim, 1992, p.43), and involve the selection by the researcher of as wide a sample as possible, recognising that it is not representative of the population. A purposive or judgement sample in a qualitative study is not subject to statistical rules regarding sampling errors, sample size and population estimates. Judgement samples are valid in cases where the true population size is impossible to determine. Judgement samples can be combined with "snowballing", where individuals identified for the survey are asked for the names of other individuals that may fit the sampling criteria.

The grounded theory approach suggests that a sufficient sample size is reached when additional interviews fail to identify significant new themes, a condition that Strauss and Corbin (1998) describe as *theoretical saturation*. A coding category is considered saturated when new concepts are no longer emerging from the data and the nature of identified categories, concepts, their properties, and theory relations are well understood and have been validated through cross-case comparison.

5.2 Data Analysis

As discussed in the preceding sections, the techniques adopted in this study were derived from the grounded theory approach to data analysis (Glaser & Strauss, 1967; Strauss & Corbin, 1998). While grounded theory provides a conceptual framework as well as high-level procedures for approaching qualitative data analysis, the work of Miles and Huberman (1994) provided detailed descriptions of techniques and tools, examples, and helpful advice on how researchers (especially novices) can apply qualitative techniques.

The grounded theory approach involves several different iterative and overlapping coding processes (Strauss & Corbin, 1998). Though the originators of the approach suggest a sequence of steps, each step is repeated as new data are collected and analysed. The first step is called *open coding* and involves generating categories and concepts, identifying properties of the concepts, and analysing how categories and concepts vary across these properties and across other dimensions. *Axial coding* is the process of further developing categories and the identification of sub-categories and the nature of their relation to parent categories. The third step is *selective coding*, the integration and refinement of the categories that have been identified. At this stage, the analyst begins to focus on categories and concepts with high "conceptual densities", i.e., patterns and themes that are recurring and that appear to relate to other categories in the emerging framework. The final step is the identification of a *central category or categories*. The central category is the main idea behind the research, what the research is essentially about. All other categories and concepts identified in the framework should relate in some way to the central category. Miles and Huberman (1994) argue that, as often as not, the decision on when to stop creating new codes is a function of budget or time constraints rather than any specific scientific principles.

Miles and Huberman (1994) classify data analysis codes along three dimensions: descriptive, interpretive, and pattern codes. Descriptive codes identify a text fragment as relating to or as an example of some phenomenon. Interpretive codes embed the analyst's understanding of a piece of text relative to the context in which it was collected. They describe the difference between descriptive and interpretive codes as public versus private in orientation. Descriptive or public codes simply tag a section

of text as an instance of a well-recognised concept. Interpretive codes include elements of the analyst's in-depth understanding of the topic area and how the private code fits into this conceptual framework. Finally, pattern codes are used to identify segments of text as examples of a theme or pattern that emerges from the data during analysis. Pattern codes represent the inferential or explanatory concepts that are identified during analysis.

According to Lofland (1971), the role of the qualitative researcher during data analysis is to "provide an explicit rendering of the structure, order, and patterns found among a set of participants." He claims identification of the characteristics, causes, and consequences of phenomena, their forms and variations, as among the central goals of qualitative analysis. To this end, Huberman and Miles (1994) describe an interactive process of qualitative data analysis that consists of three major components: data reduction, data display, and conclusion drawing/verification.

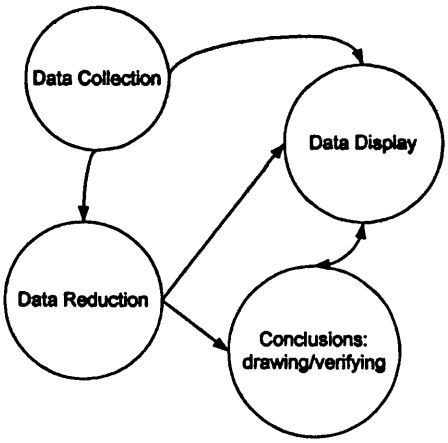


Figure 14 - Interactive Qualitative Data Analysis

Because qualitative studies making use of full-text transcripts typically result in a very large quantity of data to analyse, data reduction is used to condense this mass of data as they interact with a conceptual framework of data codes, code themes and code clusters. Data display is the process by which data are then organised and presented in a way that facilitates in-depth analysis. Drawing and verifying conclusions consists of identifying the themes and patterns in the data that are the basis of meaningful conclusions. A range of techniques and tools facilitates these processes at each stage of the analysis.

Huberman and Miles (1994) argue that theory is developed from qualitative data through the iterative analysis and identification of relationships, or a "dialogue", between the emerging theory and its conceptual framework, represented as a correspondence between the codes and the data under study. The goal of this type of analysis is not certainty, but the identification of regularities that are "probable, reasonable, and likely to be true" (Huberman & Miles, 1994, p.431). Among the specific techniques they recommend for the analysis of qualitative data are comparisons and contrasts between cases, identification of patterns or themes, clusters and linkages among the data, identification and analysis of negative cases, and identification of surprise cases.

Lofland (1971) provides a conceptual framework for data analysis arranged on a continuum from the microscopic to the macroscopic. These categories provide a basic taxonomy for the units of analysis that can be considered in a qualitative study. He defines these units of analysis as follows:

1. Acts – short term actions
2. Activities – longer term involvements of the participant
3. Meanings – verbal "productions" that define and direct action
4. Participation – participants' involvement in and adaptation to their situations
5. Relationships – interrelationships between the participants and others
6. Settings – the overall setting in which participants act

Lofland's taxonomy was helpful in the early stages of analysis when it was unclear exactly what sort of 'things' were being sought in the data. Identification of utterances that mapped to this framework, as well as explication of the characteristics and variations among each of them, was an early step in analysis.

Strauss and Corbin (1994) suggest the continual questioning of the data throughout the analysis process. The types of questions they suggest take several forms: sensitising questions, theoretical questions, structural questions, and guiding questions. Sensitising questions help the researcher to explicate concepts from the

data. Theoretical questions are meant to help build relationships between these identified concepts. Structural questions range from reviews of the developing theory, for example, which concepts are best represented, to pragmatic issues such as what direction the sampling frame should take based on the evolving theory. Guiding questions relate directly to data collection and help to evolve the questionnaire itself based on the evolving theory.

A second basic process central to the grounded theory approach is the use of comparisons applied to the data, in particular, the use of theoretical comparisons. Theoretical comparisons are employed in situations where a potential concept is identified in the data, but its relationship or strength relative to other concepts is not immediately clear. These comparisons are useful for identifying overlap of similarity with other emergent concepts, in some cases resulting in the combination, subsumption or elimination of one or another of the concepts. Comparisons are thus crucial components in the development of more abstract and generalizable theories. While quantitative, statistical studies tend to minimise the importance of deviant cases as outliers, qualitative approaches consider these cases to be an important component of the research since alternative views can often provide important insights into the phenomena under study (Frechtling & Sharp, 1997).

Although not typically provided in 'pure' qualitative research, Berg (1998) argues that code frequencies help to index, organise, and explain the results of an otherwise essentially qualitative analysis. Providing frequency or magnitudes in qualitative analyses is meant to assist with understanding of the work, they are not provided as a finding in and of themselves. In other words, the presence of twice the number text fragments with a particular code does not necessarily suggest that this concept is twice as important in the domain of study, but does suggest a potentially interesting concept worthy of further study. In particular in this study, there was no standard length to a coded text fragment, so frequencies do not help to understand the depth of a given participants commitment to the concept being discussed.

5.3 Reliability & Validity In Grounded Theory Research

Though some qualitative methodology theorists argue against traditional scientific measures of validity and reliability in research (e.g., Klein & Myers, 1999), others

hold that qualitative research must still conform to rigorous standards in their protocols and analyses (e.g., Marshall and Rossman, 1999). Much qualitative research has been subject to criticism that arises from its over-dependence on anecdotal evidence (Silverman, 1993). Among the techniques suggested for improving the credibility of qualitative research are testing alternative explanations and looking for negative cases (supported by the *constant comparison* analysis approach in grounded theory) and the use of *triangulation* both in analysis techniques and the data sources selected for sampling (Patton, 1990).

Traditional measures of the soundness of scientific research include reliability and validity. In a qualitative study with a necessarily interpretive aspect, degrees of reliability and validity are difficult to quantify. According to Pandit (1996), results are reliable when they are replicable at some level of understanding and shared meaning. He describes the types of validity as: construct validity – well specified research procedures; internal validity – relationships found in the study actually exist; external validity – findings may be generalised to the domain.

To a large extent the reliability of qualitative studies is determined by the extent to which the research documents the process by which reported results were derived (Silverman, 1993). Though the reliability of the coding process can be quantitatively determined through the use of multiple coders and measures of inter-coder reliability, but this presumes that multiple coders are available and they all share the same depth of understanding with regard to the theoretical framework being used to structure the analysis. This can be difficult in dissertation research, which rules out co-researchers and which typically has limited resources available to fund research assistants.

In qualitative research, especially that which employs coding methods and analysis techniques drawn from grounded theory, validity is enhanced through the concept of constant comparison and the continuous search for examples which contradict emerging theoretical constructs. Huberman and Miles suggest a cautious approach to the construction of generalizations from cross-case qualitative analyses (Huberman & Miles, 1994). While the selection of participants (cases) from many contexts and settings does enhance external validity, they warn against the construction of abstractions that result in a loss of resolution at the individual case level. In particular,

they argue that these high-level abstractions can lead to the diminution or loss of the "local web of causality" that is responsible for the phenomena found in a given case.

5.4 Study Design & Procedure

This section provides details on how the grounded theory approach was adapted and applied to the domain under study. The first sub-section describes how the conceptual framework that emerged from the preceding chapters was incorporated into the interview guide or questionnaire that was used in the study. The next provides an overview of the study participants, the sample. The third describes how interviews were conducted and the techniques and tools that were used. The final sub-section describes the approach taken to analysis of the data collected.

5.4.1 Conceptual Framework & Interview Guide

The study reported here made use of a pre-empirical conceptual framework derived from the literature integrated explanation facilities, theories of explanation, and design rationale as described in the previous chapters. This conceptual framework formed the basis of the interview guide, or questionnaire, that was used in the interviews as well the initial code set that was used to categorise concepts during analysis. The questionnaire included elements of a "hidden agenda" (Oppenheim, 1992), which exists when the researcher asks questions that are only indirectly relevant to the topic of interest, but are designed to elicit responses and discussions that relate to the central issues. The hidden agenda incorporated into the questionnaire relates to the efficacy of design rationale as explanatory content. Rather than explaining the design rationale approach before the interview and risk priming participants, the questionnaire instead relied on more generic questions related to their use of "information from the analysis and design phases of an IS project." Follow-up probe questions were then used explore elements of the hidden agenda, and this helped generate additional questions or concepts for further study (Oppenheim, 1992, p.52).

Sudman and Bradburn (1982) highlight the value of eliciting both the cognitive and the action aspects of the participants' responses. The former involves attempting to gauge what the participant thinks or knows about the issue introduced by the question. The latter involves determining the extent to which the participant is willing to take action to operationalise their ideas. A key element of questionnaire design is ensuring

that the questions attempt to explicate not only what participants believe or what they do, but the interplay and potential discrepancies between what they believe and what they do (Strauss & Corbin, 1998). The questionnaire developed for this study incorporated these ideas, though in general only a single question exploring either the cognitive or action elements was included for a given topic, follow on questions such as "why" or "how" were employed to query the other dimension.

Though contrary to traditional survey designs that employ quantitative analyses, Huberman and Miles (1994) claim that revision of a qualitative interview guide (the questionnaire) throughout the study actually enhances internal validity. They suggest continual adjustment of the questionnaire to help explore themes that emerge from prior interviews. In the study reported here, the interview questionnaire was first pre-tested on three graduate students who fit the general participant profile by virtue of their industry experience. This helped both to calibrate and 'debug' the questionnaire as well as the researcher's own interview technique before going out to the field. This pre-test identified several issues with the original questionnaire design including questions that proved redundant in their meaning, and questions that were confusing in their wording. The pre-test also helped to highlight several concepts that were not adequately probed by the original questions and questions that did not elicit responses in sufficient depth to be useful. Major adjustments were made to the questionnaire after this pre-test, when some questions either proved to be problematic in the field, or did not appear to elicit a response or discussion within the desired topic area.

The second stage consisted of an initial set of six interviews. These six field interviews were conducted in the metropolitan London area between 20 April 1999 and 13 May 1999. After these six interviews had been conducted, a preliminary analysis was performed on the resulting data with two goals: to further develop the initial conceptual framework and to determine whether modification of the interview question framework was required. This analysis helped to identify areas where both the questionnaire and the researcher's use of follow on and probe questions could be improved. Following this initial analysis, a series of 17 interviews were performed in the United States. During the week of 12 July through 19 July 1999, nine interviews were conducted in the San Francisco Bay area. During the week of 20 July through 27 July 1999, 8 interviews were conducted in the metropolitan Boston area. Finally,

between 25 September 1999 and 6 July 2000 an additional four interviews were conducted in the United Kingdom.

See Appendix A for the two versions of the questionnaire used in the study.

5.4.2 Study Participants

Selection of a study sample from the population of information systems professionals and information systems users was carried out with the goal of supporting a defensible claim for ecological validity in the methodology employed. Study participants were identified and selected based on their involvement in IS development, implementation, user support, and documentation. Selected participants represent a (limited) cross section of the population employed in designing, building, managing, and documenting a wide range of IS in many different application domains.

Participants represent a broad range of different information system roles from software engineers and programmers to senior management. The organisations represented range from small technology start-ups to multi-national corporations and were engaged in different industries from traditional manufacturing to custom web site development. Participants were geographically dispersed, being drawn from Great Britain (England and Scotland, with a concentration in the metropolitan London area) and the east and west coasts of the United States (metropolitan Boston and the San Francisco Bay area including Silicon Valley). The sampling method included aspects of purposeful (Patton, 1990), maximum variation (Guba & Lincoln, 1985), opportunistic, snowball, and judgement (Oppenheim, 1992) approaches. The initial set of interview participants were selected based on personal contacts including former colleagues, vendors, customers, and friends. The sample was expanded from its initial base by asking participants whether they would be willing to put forward the contact details of other systems professionals that might agree to be interviewed.

A note on confidentiality. Although a full list of participants appears in Appendix B, in the results section individual participants are not identified in relation to concepts that emerged from the analysis. Where verbatim quotes are provided to exemplify, emphasise, or highlight a particular point, these quotes are also anonymous ensure confidentiality.

5.4.3 Interview Techniques & Tools

It has been said that the quality of the information obtained in a semi-structured, qualitative interview is largely a function of the researcher's skills and personality (Patton, 1990). While the latter factor was accepted as is in this study, many steps were taken to address the former given the limited experience of the researcher. The standard set of interview tools consisted of a contact sheet, a copy of the interview guide (questionnaire), a pen, a standard cassette recorder and microphone, and two blank, 90 minute cassette tapes. The contact sheet was used to note the person, time, and place of the interview as well as to record any particularly salient points that emerged of the interview immediately after it was conducted. The interview guide was used both to structure the interview and to record highlights of the interview.

Though some researchers argue against the use of tape recorders in interviews on the grounds of the intrusiveness of the technology (Guba & Lincoln, 1985), use of a tape recorder allows the researcher to concentrate fully on the discussion and facilitates asking follow on and probing questions as warranted by the discussion. All of the interviews were taped with the unfortunate exception of one (very high-profile) participant who could only be engaged over lunch in an extremely crowded and noisy restaurant in San Francisco. In this case detailed notes were taken of the participant's responses. A standard, auto-reverse cassette tape recorder with a separate, omni-directional microphone was used for all other interviews. In general, the interview settings were of high quality and conducive to uninterrupted conversations. Most were performed in conference rooms or personal office space though some took place outside and others in cafes or restaurants. Participants were generally extremely conscientious in avoiding or minimising disruptions such as phone calls and other interruptions. All of the interview participants were native English speakers.

5.4.4 Data Analysis Techniques & Tools

The quantity and potential complexity of the data that were collected for the study suggested that the process would benefit from computer-based analysis aids. A range of qualitative data analysis support tools was considered for assisting with the analysis. The qualitative data analysis packages NU*DIST (www.qsr.com.au) and Atlas/ti (www.atlasti.de) were the candidates most closely considered based on the fact that they are licensed by the School and are generally well regarded by colleagues

and faculty. Based on reviews and feature comparisons of these competing packages (Miles & Weitzman, 1994; Weitzman & Miles, 1994), and fairly short hands-on evaluations, Atlas/ti was selected for the initial analysis of the first six interviews. The key factors in this decision were the well-designed, integrated-function user interface of the software, the ability to code text passages of arbitrary, rather than a pre-specified size (e.g., word, line, sentence, paragraph), and a feature that supports building non-hierarchical networks from the code sets (NU*DIST supports construction of only hierarchical networks). Atlas/ti proved very effective in the initial data analysis phase, it is a well-designed, stable, and easy to use package, and so was selected for the remainder of the study.

5.5 Results

This section reports the results obtained from the study. The results are organised into two major sections. The first section, Explanation Content, reports on how study participants described the nature and content of IS explanations. The second section, Design Rationale as Explanation, discusses the major themes that emerged from responses related to the explanatory utility of information gathered from the analysis and design phases of IS development projects. This section is further sub-divided into Factors, describing what emerged as the major factors that impact the process of capturing design rationale explanations, and Outcomes, describing what participants identified as the most significant effects that result from the use of analysis and design information, in the form of design rationale, as the basis for explanations.

5.5.1 Explanation Content

This section reports on the most significant findings relative to explanation content in the information systems domain. Since participants frequently described IS explanation content in terms of the questions that might be asked in the context of IS use (e.g., how does it work?), these questions are reviewed before the analysis turns to the central issue of the content most suited to answering *why* questions. The following table provides a summary of the finding related to explanation content. The Density values, e.g., 66/23, refer to the total number of coded text fragments (66) and the number of interviews in which the code appears (23).

Table 3 - Summary of Findings Related to Explanation Content

Explanation Content Type	Density	Definition and Relation to Theory
Operational Explanations		
How do I use it?	66/23	Step-by-step instructions. Most closely related to the functional theory of explanation, though the purpose of the instruction sequence was not always made explicit.
What does it do?	35/18	The fact that a given feature exists, its identity or name, and its definition. Though a basic building block of any explanation, what explanations relate most closely to the relevance relations from the pragmatic theory of explanation. It is assumed that the level at which a system object is identified and defined is based on its relevance to the explanation being provided.
How does it work?	55/24	A more complex question that involves providing information about the structure and function of system features. Most closely related to the functional theory of explanation in that the question may be reformulated as "How does it work to achieve my goal?"
Why Explanations		
D-N Explanation	26/13	Factors that function as laws to constrain the design choices available in a particular problem-solving context. Sometimes relate to a relaxed view of the laws central to D-N explanations, in other cases this relationship is tenuous.
Pragmatic Explanation	66/24	Explanation content depends on one or more of several different factors including attributes of the end user, of the system, and of the specific task.
Functional Explanation	33/15	IS explanations relate to the purpose of the system or one of its features. The types of purpose discussed included those at a micro level, related to a particular feature, and those at a macro, exogenous level that relate the system or feature to its purpose relative to the business or other context in which it is used.
Rational Choice Explanation	25/17	Explanations that draw upon the decision making process followed by the IS design team, how alternatives were identified, deliberated, and finally decided upon are the class of explanation that intuitively are most likely to benefit from access to design rationale.

Operational Explanations

The types of explanation content that emerged from the study may be examined from the perspective of the explanation seeking questions to which they respond. At the most fundamental level these questions are related are as shown in the following diagram:

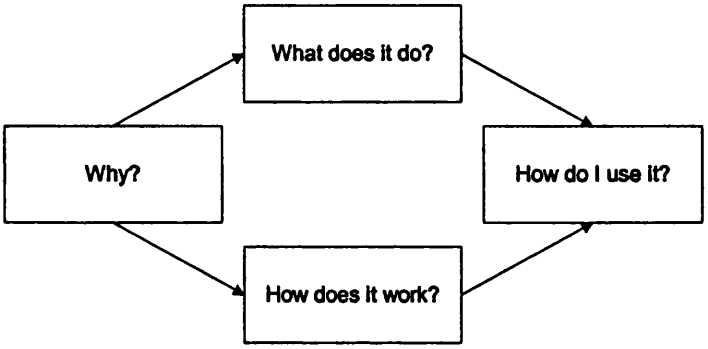


Figure 15 - Basic IS Explanation Seeking Questions

How do I use it?

How explanations were the type most frequently mentioned in response to interview questions related to IS explanations. Providing explanations in response to a *how* question was by far the most frequently cited type of explanation content with 66 coded text fragments in 23 of the 27 interviews. A *how* explanation was typically described as a step-by-step, "cookbook" approach to the use of a given system or system feature. These types of explanations were often considered the most practical given the time and interest constraints of IS users. *How* explanations without explicit reference to goals present something of a conundrum both because of their conceptual density in the results and because they do not address the central explanation question of *why*. Participants sometimes acknowledged the lack of explanatory depth provided by *how* explanations, for example:

> *"...we would go with more of a step by step on how to use a feature rather than an explanation of what it is and what it's doing."*

How explanations are most closely related to the task-based model of instructional technology described in Chapter 2. *How* explanations are seen as especially important for new users of a given IS. This factor relates to the practical nature of these explanations, new users typically need to become productive quickly with a system and providing how explanations is seen as the most efficient way of accomplishing

this goal. *How* explanations are often considered a 'first line' explanation approach that needs to be supported with more detailed information on what a feature does, and why it works in a certain way. One participant put it this way:

> *"And then there are several levels of information. At the top level, here's what you do to the system to get what you want done for you, what are the mouse clicks in and so forth and so on. Second level is what is the value in the results that you get? You know, what is the value added of the results? What does it do for you? How does it solve the problem? Okay? And then the last level is how does the system do what it does? Right? What's the technology behind it right? Why does it work? Okay. So those are the different levels I think providing people with the package. Some people won't go beyond the first couple and there's only a few people will go all the way down to the implementation technology details."*

Answers to *how* questions are most closely related to the functional theory of explanation in that an instruction sequence always relates to some end goal or purpose, even if this goal or purpose is not always made explicit. However, a questions remains whether instructions can be characterised as explanations, in the sense of providing deep understanding, without an explicit purpose being provided for the task. Answering the question *"how do I use it?"* does not involve the same content as the answer to the question *"why do I use it that way?"*

What does it do?

What does it do? explanation seeking questions occupy a second tier, along with *How does it work?,* in terms of their theoretical density. Providing answers to *what does it do?* questions was cited in 35 text fragments in 18 separate transcripts. These explanations consist of several dimensions including existence - the fact that a system or system feature is actually there, identity - the system or feature's name or reference, and a definition - the contents of which can vary widely and begin to overlap with other explanation content types. Simply identifying the existence of a system feature was seen as an important, if basic, element of an explanation in many cases. As one participant put it:

> *"Every programmer's dream is that the people using the software'll love it so much they'll try to find everything you put in it. But that's pretty rare."*

In terms of their relation to the instructional technologies discussed in Chapter 2, these explanations map most closely to the system or software model, where a system

is explained by providing an exhaustive list of the windows, fields, menus, buttons, and other objects that make up its structure. As is the case with *how does it work?* explanations, *what does it do?* explanations are seen to be especially important for new or novice users of a system who may not understand that a given feature is part of the standard set common to a particular type of application, for example, the insert table command in a word processor or the lasso tool in a drawing program.

Definitions are important as they represent the established meaning of a given term as used in a system. One issue with definitions is that the meaning assigned to a given term during the analysis and design phases of a project does not persist in the use context. As one participant put it:

> *"I have always hoped for the information that's captured up front in models like definitions and things like that, to make their way into on line help of the end system. But I'd say it's been difficult to carry it all that way through."*

This issue is exacerbated for *compound* terms, for example, a field value that is calculated from more primitive data elements. For example, an explanation of an invoice line total that is calculated from the quantity sold, the sales price, and additional tax and shipping charges may potentially need to explain each of these more basic elements. Another problem with maintaining and communicating definitions is that system objects are often given two different labels, the label the user sees and the label used for the underlying system object. For example, due to naming format restrictions for the schema objects in several database management systems (DBMS), these names may not include spaces while the label used to identify the same object in the user interface form employs a more 'friendly' label that makes use of spaces. This means that those with the most in depth knowledge of what a given object really is may use a different name than those to whom the object is most important, the system user.

Though a basic building block of any explanation type, *what does it do?* explanations relate most closely to the relevance relations from the pragmatic theory of explanation. It is assumed that the level at which a system object is identified and defined is based on its relevance to the explanation being provided.

How does it work?

Answers to questions about *how a system works* represents a more complex set of explanatory content than those required by either the *how do I use it?* or *what does it do?* questions discussed in the previous sections. Without the supporting information supplied with an answer to a *why* question, answers to *how does it work* questions rely on information describing system processes, the information flows between system components, and the flows between different systems. *How does it work* explanations were cited in 55 text fragments on 23 different transcripts. This information is characterised by its general invisibility to the end user, it describes what is happening in the background when the system features made apparent by *what does it do?* and *how do I use it?* explanations are put to use.

Providing an understanding of the interrelationships between different system components helps users to understand how a particular component is used, what the 'upstream' requirements are for using the component, and the 'downstream' effects of using the component in a certain way. For example, the system might identify the source of some piece of information that is being used by the current feature or process as well as describe how the information that results from the current transformation will affect subsequent features and/or processes in other parts of the system. Understanding how the system components interact represents a step towards more effective use of the system overall, for example:

> *"And so here's my task, here's what this, here's what this tool really does, and here's how this tool is put together so now I know how to use it in other places. That's what I guess would be most complete."*

Because explanations in response to *how does it work?* questions provide information about the causal dependencies within a system process or between system modules or independent systems, this information may also be used as a component of a *why* explanation. This will be discussed further in the next section.

Why Explanations

As shown in Figure 15, answers to *What does it do?* and *How does it work?* questions about an information system are supported by information capable of answering questions about *why* certain features exist and *why* the system works the way it does. Some participants expressed the view that end users were often insulated from the reasons behind why an IS is delivered with a particular feature set, and one key

promise of the DREX is that it goes beyond *what* and *how* information to give the rationale behind system architecture, features, and context. As discussed in Chapter 3, philosophical theories of explanation purport to provide frameworks for answering these why questions. The following sections discuss the central concepts related to why explanations that emerged from the study in terms of the philosophical theory to which they most closely relate and the manner in which they are applied in the context of IS use.

Deductive-Nomological Explanations

As discussed in Chapter 3, deductive-nomological explanations are those that rely upon the existence of laws. The nature of a law, however, is a point of controversy, especially as the subject domain moves from the physical sciences to the social sciences, or, in this case, the socio-technical sciences. In only very few cases did study participants refer to strict physical laws as relevant to the explanation of an IS. Though such cases did emerge, a far more prevalent theme was the role of various types of constraints that act to bound IS design alternatives, and may therefore be important in an IS explanation.

> *"Well, yeah, I think that the analysis and design aspect of the project obviously starts with you know a specification of the objectives and the constraints on the project, okay? That's kind of what bounds the project and stating that clearly as part of the overview in the end users documentation could be helpful, saying this product is designed to do X, Y and Z and not A, B and C because of C,D and E. It could give a perspective on what's happening."*

The nature of IS design constraints spans a broad range of types, from those that stem from physical laws, for example those that bound computer and network performance and capacity, to constraints that are properly seen as social constructions, including generalisations such as industry standards, accepted IS development practices and methodologies, and the norms of behaviour that guide IS development teams. An IS development project may also be constrained by second order factors within the domain of the IS. An additional type of constraint that emerged was that implied by the high-level system architecture model, which serves as a frame or scaffold into which subsequent system features and process are designed to fit.

Domain Constraints as Laws

The constraints that bound an IS design within a particular domain emerged as an important component in explanation. In the one case where one of the participant's system projects actually includes an explanation capability, this capability is was constraint-based. As the participant described it:

> *"And we find that the explanation capability is still very valuable even in the case where you only have five or six constraints."*

In general, constraints were closest to the more relaxed notion of laws that Hempel admitted was required to support application of the D-N model in the social sciences (see Chapter 3). For example, legal requirements sometimes determine how a system will be used as in:

> *"And also what my role is as a user of that system in the form of either data integrity and any other legal requirements that might come along."*

Other domain constraints included the security standards that determine user access to certain features.

In many cases the nature of the constraints discussed were difficult to classify as laws, even in the more relaxed view allowed by the D-N theory as applied in the social sciences. For example, budget and time constraints on development projects are always a constraint but it is not always clear how these can be subsumed under laws except in the sense that both of these resources are never infinite.

Technical Constraints as Laws

Some participants discussed the role of technical constraints, for example, the constraints introduced by one or more elements of the technical platform, for example the choice of PC or Macintosh, or the supported network protocol, were cited as potentially explanatory when these factors impacted design decisions that were then apparent in the 'look and feel' of the system. One example referred to the effect that the relational data model may have on the functionality of a system. Data models may be designed to support rapid data capture or rapid data access, but generally not both in a model of any complexity. How these factors are weighed and how they manifest themselves in the design of the IS is information that can contribute to a system explanation.

"They want to say oh we don't need to know about that, that's technology. But I don't agree. I think that's the foundation of them understanding actually how they're going to use the technology and would help them to trouble shoot things."

In some cases the types of laws that were cited as potentially explanatory were physical as in the strictest interpretation of the D-N model. For example, processing and read/write speed, storage requirements, and other physical attributes of the computing device sometimes bound the features that are supported by a given IS, or the frequency with which they can be used. For example:

"Constraints was a big thing that we would explain. In fact, as I just mentioned I guess, two out of three of my examples were a case of explaining the constraints on the design. It's sort of the opposite of, as I said earlier, software can do anything. Yeah, but it's on hardware."

One participant described the constraints imposed on systems that operate on a personal digital assistant (PDA) as central to any explanation of the software. These constraints include screen space, power consumption (and therefore processing speed), and wireless bandwidth as highly determinant of the kinds of IS that could realistically run on the platform.

In other cases, participants discussed the explanatory role of constraints derived directly from experimentation and empirical testing, for example on the processing performance or human-computer interface attributes of an IS. Several participants felt that communicating how the results of such tests were translated into constraints would assist end users in developing a deeper appreciation and understanding of a system, and demonstrate to them how different factors were balanced by the development team in the design process.

The System Architecture as an Evolving Constraint Base

Several participants reported that providing information about the system model or metaphor as part of an IS explanation might play a role in fostering understanding of the system's constituent components and how they interact. Explanation content that draws on the system developers' architectural model was reported as one of the more important aids to system understanding. The system model acts as kind of law in that it affects subsequent design decisions that are forced to fit the model. Architectural models provide end users with an organising framework within which specific system

features and processes can be explained by reference to how they fit within this framework. Architectural models may take the form of an organising metaphor, such as a cheque book and ledger for a personal finance application or a canvass and palette of tools, colours, and textures in a paint program. Explanations that reference a system model help close the gap between the designers conception of the system and the end users conception of the task domain that it supports and help users to understand the dependencies that exist between system components relative to the model.

> *"I would bet that you'll get some users who will just you know turn off when you, as soon as you start saying big picture but I think it would be a great help to others. I think it would, and again I think it would help the user developer relation as well because I think it would help them realise that there is this big picture and some of the decisions that are made which they think are just against them are actually affecting lots of different facets of the application."*

One participant described the system architectural model as the essential "spirit" of the system.

> *"To give the user an understanding, I believe there is such a thing as the spirit of a programme or the design ideal of a programme and if you use a programme in the spirit of the programme, things tend to go right."*

How to define and communicate the system architectural model is not always clear. In some cases participants suggested a graphical depiction of the architecture, in others participants suggested relating the system architecture to its underlying metaphor, such as the page and tool palette in many word processing application, or even to another system with similar functionality (e.g., explaining Microsoft Word to someone who uses WordPerfect using examples derived from WordPerfect's feature set).

> *"Um, I think that as complicated as it might be, sort of overall system overview of the application would just do wonders and you know, many of the projects I've been on each of the users perhaps only works with a particular facet and what frequently happens in specialist cases is the user only sees issues, or doesn't quite get it or doesn't like how the interface works cause it doesn't work well with his or her facet or his or her module that they're working on or whatever. And it certainly, I mean I'm just wondering about both the issue of making them understand the globalness of this application as well as perhaps just giving them a better understanding, I, though it's never been tried, I always wonder, boy, if we could just get the users and say okay, look, we're going to, here are a few other elements and phases of this*

application that you're never going to use but we want to give you the big picture and we want you to understand you know, that here's your piece but here's how your piece fits in with everything else."

System Architecture information provides details of the physical and logical system design and how specific components have been selected and assembled to construct the application. Also included is information on how the system operates within the local and wide area IT infrastructure, for example, its reliance on networking hardware and software or its integration with a database management system (DBMS).

Pragmatic System Explanations

Though not a distinct type of explanation content, one of the most dense themes that emerged from the study was the idea that what constitutes an explanation of an IS depends on one or more attributes of the user, the system, and the task. As described in Chapter 3, the pragmatic theory of explanation defines the structure of an ideal explanation as wholly dependent on the context in which the explanation is requested. The results of this study supported this idea, not least by the breadth of explanatory content that was discussed and the pervasive theme of explanatory content as dependent upon the nature and current interest of the individual system user.

Very nearly all participants felt that what constitutes an explanation in the IS context is dependent on the explanation requestor, the purpose for the request, and other factors. Explanation requestors, or users were classified along several dimensions including:

- Level of system expertise – novice to expert
- Level of domain expertise – novice to expert
- Level of general technical expertise – novice to expert
- Role relative to the system – user, manager, implementer, programmer, support, etc.
- Organisational seniority – Executive to manager to individual contributor
- Learning preferences – user manuals, training, one-on-one support, etc.

- Geographic location – in particular for global systems

Many participants referred to a dichotomy of user types, which may in fact represent several dichotomies. Explicating the exact structure of this dichotomy was elusive, but the following passage is representative of the general theme:

> *"I think it's more the difference between the people who are actually actively interested I making their job easier by using the tool every day. And then the people who just see it as a tool like a phone, who don't, it's the difference between the type of people that actually take the time to put speed dial phone numbers in their telephone and people who just know that they bought this new phone, it looks good, it works, I don't know what all those other buttons do and I don't care."*

Whether the different user types relates to interest level, motivation, inquisitiveness, intelligence or some other factor is not clear. What is striking however, is how often it was discussed and how emphatic some participants were regarding the existence of this dichotomy, for example:

> *"...and you know it's funny, most people fall into you know either of those categories. It's not, I haven't seen a lot of people at least around here, that are just you know middle of the road."*

There was a distinct evaluative component to this dichotomy. While in a few cases a general disdain for the performance level of end users was evident, many more participants suggested the existence of a minority of 'good' users, and users that are less 'good'. An interesting factor related to this dichotomy is that users classified as already 'good' are those that require and make use of *more* information about a system. As one participant put it:

> *"Sophisticated users usually want and need to know as much as possible."*

While for the user type at the other end of the spectrum:

> *"How do I use this? How do I do my job? How do I get home and relax and forget about all this stuff? I'd say for most people that's the most important but the other stuff for people who've got time or an interest or a need..."*

Other determinants of pragmatic explanation content included system type and the domain in which it is used. Of particular importance was the degree of configuration or customisation supported or required by the system. For distributed, complex systems such as enterprise resource planning systems, the type and content of

information required might depend on the department and business operations function of the end user.

Relevance Relations

Matching the content of an explanation to the context and interests of the requesting user appears crucial to acceptance of system-provided explanation. As discussed in the Factors section later in this chapter, many participants felt that the majority of system users have limited interest and limited time available to sort through masses of analysis and design information that may or may not be relevant to their current information needs. As discussed in Chapter 2, attempts to model the characteristics of individual end users and to tailor explanatory content to this model have met with limited success.

Contrast Classes

A fairly small number of participants (8 coded text fragments in 6 transcripts) identified contrast classes (described in Chapter 3) as an important element of explanatory content. Contrast classes explain by describing how a system feature or process "might be otherwise". In some reported cases contrast classes explain by showing how a particular feature meets some essential domain criteria, such as performance or ease of use, better than alternative designs. This type of explanatory content appears to be particularly important when the design represents a compromise solution and where a clear and more desirable design is apparent to the end user.

> *"You know every piece of user functionality can't be perfect every time you know. Software isn't magic. You have to make some compromises. You have to say, well the critical part of the system has to do this and you have, at the end of the day, the system has to be at its best in these circumstances. Like if there's a storm. In our industry, if there's a lightening storm and the system is under a lot of strain because we get lots of alarms then, and there's lots of customers they're supplying and the control engineers are going crazy, at that point the system has to perform at its best."*

Constraints of money and time were mentioned several times as some of the more important criteria that determine the form of the system and that force compromises in its design. In one case this was described as very important when end users were being forced to migrate from an existing legacy system that had been constructed to their specifications and to which the organisation had adapted over time, but that was extremely expensive to support and maintain. In this case describing how the cost

savings accrued from the use of a more standardised, perhaps purchased system would contribute to some other factor such as the level of support the users could expect was seen as important in facilitating this type of change.

Functional Explanations

Of the explanation types that explicitly answer the question *why*, functional explanations were the most common theme that emerged from the study, though differentiating *how* explanations from true functional explanation was sometimes difficult during the analysis. A *how* explanation approaches the structure of a functional explanation, for example, if task steps are given in reference to some end goal or purpose. As discussed in Chapter 3, true functional explanations relate the structure of an IS to the purpose of that structure within the system domain.

The functional explanations reported took on two forms: one relating to a 'micro' purpose and one to a higher level or 'macro' purpose. Micro-purpose explanations relate to the immediate or local goal of a system feature and/or process, without reference to the exogenous need for the feature. For example

> *"Then an explanation starts probably by stating a problem or by stating a goal like oh a lot of users have trouble removing files who's name start with a dot..."*

Macro-purpose functional explanations provide a much broader perspective on the role of a system within an organisation, attempting to rationalise the existence of an IS or particular features by reference to the goals of the corporation or institution. For example:

> *"I often find that the users like to know how the information that they're either putting in or processing is actually used. Not always but often because it gives them a sense of feeling and purpose for what it is their role is for the use of that system. And a sense of feeling and purpose again for me is a very important context setting for the user. Why are you important can often be translated into you know the firm the company value your input and that value input is listening to you and valuing your views and perceptions about that system."*

An interesting facet of functional explanations is the relationship between the micro and the macro-purpose perspectives in the sense that this relationship provides information about the relationship between the strategy behind a system project, and the structure that has been implemented to support that strategy. This information

provides the direct link between a system's functionality, the task designs built into the system, and the business or other type of problem the system is meant to address. The strength of this link helps to establish an IS as a component of a more general strategy rather than as a demonstration of technical prowess on the part of the development team. As on participant put it:

> *"Um, I think one thing overall is that, I think too many information systems projects exist for the benefit of the technology itself and not the impact on the business. I think if projects started, if more projects were started with a true business case, a sense of how it impacts the company, you know, knowledge of the return on investment, knowledge of why they're doing what they're doing and they have the users involved more up front so they understood what the overall intent of the application was, I think that would be a benefit to the usage of the system downstream."*

Business and Other Processes Supported by the System

One way in which a why explanation may be provided is to relate the operation of the system to the domain process that it supports. Explanations of this type help make the function of the system tangible and relevant to the user. Of particular benefit with this kind of explanation is that it exposes the process model employed by the system developers in their design. This can help instil confidence in the system users in that they see how the designers understood their task.

> *"Well they've got to relate their job to the system, right? So a lot of the content would have to be regarding the job not the system."*

Explanations may also be provided that draw on information related to the business case or justification for a system or system feature. This type of information helps users to understand the system in the context of the business case or high-level requirement that it supports.

Use Scenarios Supported by the System

As with explanations that draw on information about the business case that motivates a particular system development project and resulting system structure, some participants identified the *use scenarios* that were used to guide the design process as a central component in explanations of the resulting system. Use scenarios also serve as examples that map the business or other process supported by the system to the system process that supports it. Explanation content based on use scenarios may also evolve along with the context in which the system is used. This happens when users

request a 'solution' to a particular use scenario via the support organisation responsible for the system. Assuming that these new use scenarios and their solutions could be incorporated into the explanation knowledge base, this approach would allow the system's explanatory capabilities to escape obsolescence as the context in which the system is used undergoes change.

> *"So there we would give, we would give the explanation to the user of why all these features were in one place and how they related to each other by reference to what we saw as a, if you will, the crucial use scenario, namely moving a lot."*

Rational Choice Explanation

Rational choice explanations attempt to explain human actions as the product of deliberate and utility-driven decision processes. Explanations that draw upon the decision making process followed by the IS design team, how alternatives were identified, weighed, deliberated, and finally decided relative to the goals of the organisation and individual stakeholders are the class of explanation that intuitively are most likely to benefit from access to design rationale. Participants' assessments of the utility of this information was mixed and in many cases contradictory. Though in some cases participants argued that system end users would *not* be interested in the content of design deliberations, in others some of the information identified as important to the user community is clearly a product of the design team's work.

> *"That we both understand what the workflow is, how to make the process the most efficient that we have considered all the technology options out there and we've chosen the one that solves the most important capabilities and first and leave out the less important capabilities."*

Because design deliberations and resulting decisions provide information on how a system structure emerges from an analysis of the domain relative to an identified purpose, these deliberations may help close the information gap (from Clancey, 1984), discussed in Chapter 2, between the system strategy and its structure.

> *"Um, well at the moment we're looking at implementing certain types of genetic algorithms, combinatorics, stuff and some of the implementation details, there's a big crossover between that and the business domain, to throw up some of the design decisions that have been taken and put them out in the*

user domain certainly could be useful for other people that weren't present, at the initial meetings So they could say ah right, that's why they're doing it."

Though clearly not all design deliberations are interesting and useful to all system stakeholders (this point is elaborated in the sections that follow), one motivation behind this research is to identify what information from analysis and design, in the form of design rationale, is most relevant to explanations in the use context. Towards this end, the discussion section that ends this chapter will attempt to integrate these results into the framework developed in earlier chapters.

"I think that for modern software projects, one of the things that needs to be decided up front is a sufficiently granular priority system where features are prioritised. And people talk about must-haves, nice-to-haves, that sort of stuff, something slightly more expressive than that is useful."

5.5.2 Design Rationale as Explanation

In analysing the data collected from those parts of the interviews that speak directly to the concept of using what was generically termed analysis and design information, and specifically termed design rationale information, the themes that emerged were organised into two categories: the generally negative factors that make capturing design rationale difficult, impossible, or not worth the effort, and the generally positive outcomes that result assuming this information has been captured and then put to use in the construction of explanations. In other words, if the analysis and design information could be captured efficiently, and the information designed in such a way as to easy to retrieve when needed, then access to this information would be useful. Negative factors and positive outcomes were roughly equivalent in their theoretical densities, making a clear determination of whether the approach is viable difficult. The following sections discuss the most common themes that emerged from this aspect of the analysis and these are examined further in the discussion section that follows.

Table 4 - Summary Factors and Effects of DR Explanations

Factor/Effect	Density Codes/ Cases N = 27	Comments
Factors		
Lack of interest	32/18 Negative	End users are not interested in information from the analysis and design phases of an IS project. An IS is simply a tool to accomplish some task, a deep understanding of the rationale behind the development of these tools is not necessary.
Champions	17/12 Positive	Both a factor and an effect, champions are users with special systems and/or domain expertise that may be the among the strongest proponents and beneficiaries of the DR explanation approach.
Lack of Time	11/8 Negative	End users and other IS project stakeholders simply do not have the time to access and design rationale-based explanations.
Design by Committee	8/6 Negative	Otherwise known as the 'too many cooks' factor. Design by committee frequently leads to over analysis and a corresponding lack of design decision making, resulting in longer development lead times.
Information Translation	4/4 Negative	Information from the analysis and design phases of IS development projects would require significant translation to be made meaningful to most end users.
Competition	4/3 Negative	Providing too much of the design information behind a commercial IS may compromise the competitive position of the organisation.
Outcomes		
Participation	35/19 Positive	Access to the design rationale behind an IS may promote a culture of participatory design.
Communication	29/14 Positive	A mechanism to make the deign rationale behind an IS public would serve to communicate more clearly the purpose, structure, and operation of the system.
Organisational Memory	24/14 Positive	A design rationale-based explanation tool would serve as an organisational memory of both the IS, and the domain in which it is designed to be used.
Credibility	21/11 Positive	Exposing the design rationale behind an IS may aid acceptance of the system and increase the credibility of the IS organisation.
Design Decisions	16/10 Positive	A public design rationale would serve as a basis for more informed design decisions.
Customisation	6/2 Positive	Design rationale helps to communicate the cost-benefit structure that underlies how features are prioritised, as well as the dependencies and constraints that impact the viability of new feature requests.

Factor/Effect	Density Codes/ Cases N = 27	Comments
Requirements Tracing	5/4 Positive	Access to the design rationale would help project stakeholders to understand how domain requirements were realised (or not) in the resulting system.

Factors

The following sub-section describe the various factors that emerged from the study relative to design rationale capture and use. These factors are generally negative in that they constitute reasons why it may not be practically possible to capture the amount and types of information prescribed by the design rationale approach, or even if captured, this information may not be useful to IS project stakeholders.

Lack of Interest

A pervasive theme that emerged from the study was that participants believed many users generally have little interest in explanations derived from the information generated during the analysis and design phases of an IS development project. According to many participants, the process by which IS requirements, which are of interest, are translated into a design involves too much 'technical' detail to be of any utility in the context of system use.

> *"I actually think there's plenty of stuff that they don't necessarily need to see. I think you know larger applications that are highly distributed, they have very evolved architectures, that you know have lots and lots of intricate detail behind them, is almost meaningless to an end user and it ultimately doesn't affect their usage of the system."*

In many cases participants felt that system end users view the systems that they use in their jobs as simply tools, and that the inner workings of these tools are of little import.

> *"I don't know. I'm just trying to think well if you're a user you know the system is not is not the end all and be all of your job. The system is a tool to get your job done. Therefore the quicker you can use that tool to get your job done the better."*

In some cases participants expressed their belief that the system design process was uninteresting, and that what matters most are the features obtained in the resulting system.

> *"So they're more interested in terms of you know functionality and capability as opposed to how we arrive at those functionalities and capabilities."*

Exceptions were made in the case of open source software, where it was argued that access to design information and the inner workings of these software tools is at the core of this ethos.

> *"But I would still sort of say that probably means 90-95% of the decisions that are being made during the development process won't need explanation."*

Lack of Time

Even for users that are potentially interested in more detail about the systems they use, constraints of time may often prevent them from acting on this interest. As one participant pointed out, even stakeholders close to an IS development project frequently do not read all of the documentation presented to them.

> *"I found that to be a problem, to get people to really read documents. The development team to read what was written in a spec. So users, that's even harder I think to get them to read and bless a spec. They really don't want to seem to want to spend that time looking at reading and making sure that it captures what they are looking for."*

Part of the problem might be with the format of most IS design documentation, relative to the context of system use. In situations where the end users is attempting to determine the meaning of some system process or feature in order to perform a task, there may well be more motivation to see what is relevant from the design, without needing to read all of the design documentation out of context.

> *"I would say they're getting what they need because again, people don't have time to read stuff that's not impacting them right now."*

Design by Committee

Over-participation by the end user community can result in IS development projects that fail to orient around a clear statement of purpose, the essential system model or metaphor becomes diluted by many compromises as the design team attempts to incorporate a wide variety of design inputs and feedback. By opening up a design to

scrutiny and accepting input and feedback to the design process, developers establish an implied contract that says if users provide suggestions, those suggestions will be incorporated into the design. When this does not happen for whatever reason, end users may feel that their input was ignored.

> *"One of the things, one of the challenges that you have is actually getting an agreed and signed-off set of requirements. Because the perception often is that well you've got all these different users and they can't agree on what they want and a-ba-ba-ba-ba and so you might say, you go to this set of users and they say, yeah well this is what we want, we don't want that. And then you go to another set of users and they'll tell you the complete opposite thing. So often you're left with the unenviable task of trying to reconcile the requirements of lots of fairly disparate user groups in different you know organisations."*

Participation can also impede the design process as each decision needs to be justified to an ever-larger group of stakeholder.

> *"We had to make a decision, we understand it's probably not the best one but you know there are reasons why we've gone that way. And having to constantly explain those design decisions to everybody who wants to know is going to slow it down."*

Of course, one of the goals of the DR explanation approach is to provide a common communications medium for design decisions, as well as a mechanism for gathering design input and feedback.

Champions

As discussed in the section on Pragmatic Explanation above, participants identified a dichotomy of end user types, characterised by a clear difference in their technological or systems aptitude and interest. Those with a strong interest are sometimes referred Champions are system users with a special aptitude, interest, or motivation related to the applications used by their company, department, or other organisation. These individuals are the 'power users' or 'gurus' to whom others rely and defer for special IS knowledge. In some cases their role is formalised, for example, as subject matter experts responsible for ongoing local support for a system. Often these individuals are involved in the development and/or implementation of a system as user representatives, and this special responsibility persists across the life-cycle of a given system, unless they decide to leave the organisation, which presents new problems related to organisational memory and knowledge transfer.

"...they're the ones who you actually might have to spend some time satisfying their curiosity about the building blocks of the system and about the design cause they are genuinely more interested which is okay."

In some cases participants expressed their belief that only these champions would be interested in the product of design rationale as explanatory content, but in many cases it was felt that this information would play an important part in supporting and developing the expertise of these individuals.

Information Translation

Providing information from the analysis and design phases of an IS development project would entail the additional overhead of formatting and designing the information in such a way as to be tractable by end users unused to form and formalisms often used to represent designs. This translation process involves both condensing and annotating this information so that 'lay' users can interpret its meaning.

"I think you have to decide what the users need to be able to do and for instance, the user may, most typical users may not need to know nitty gritty details of how to use a particular menu command and down six levels or something, maybe you could mention that it's there but the problem with giving all this information is that it can be overwhelming to somebody. You have to scope it down to the place where people will actually read it. If the thing is too long or too detailed, a lot of people just will throw up their hands and not even try."

Competition

The competitive nature of the IT industry might result in companies being reluctant to expose details of their products functionality, especially their weaknesses, to a base of users that might include their competition. This issue highlights an important issue with access to design rationale information, the security structure that may need to be established to limit access to 'authorised' use.

"We do think long and hard on the way we solve certain problems and I'm happy we've got some very very neat solutions. I wouldn't want to expose the blueprint to anybody who might be a competitor."

Outcomes

The following sub-sections describe the outcomes participants suggested were most likely to accrue from the use of design rationale-based explanations. In contrast to the

factors identified in the previous sections, which are seen to impede design rationale capture and use, these outcomes are positive in their perceived effect on system usability and organisational fit. This apparent contradiction will be explored further in the discussion section at the end of this chapter.

Participation

> *"The thing they hate most is suddenly receiving something on their desk which they've not necessarily been involved in at the design stage. But again what I often see is the technicians making assumptions about design in the look and feel."*

One of the most promising outcomes of the design rationale-based explanation approach is the potential to provide a medium for enhanced participation, call it virtual participation, in the IS design process. The advantages of participatory design are well-documented, as discussed in the last chapter, and multi-faceted. Among the benefits participants in this study discussed were better design decisions; synchronised, continuous feedback to the process of IS design; and increased accountability on the part of those stakeholders involved in the design process.

> *"So someone from the user side should be involved in making sure that your analysis of the business problem stays on track with their real problem."*

Participation in the design process helps to ground design decisions in the human and organisational context in which the system will be used. However, it is important that approaches to increased participation not be seen as only symbolic or superficial. Users need to feel that there input is valuable, even if it is not adopted into the actual design. By exposing the design rationale and including user-generated design questions, options, and criteria, they are able to examine how there input and feedback has been incorporated into the design process.

> *"I often find that the users like to know how the information that they're either putting in or processing is actually used. Not always but often because it gives them a sense of feeling and purpose for what it is their role is for the use of that system. And a sense of feeling and purpose again for me is a very important context setting for the user. Why are you important can often be translated into you know the firm the company value your input and that value input is listening to you and valuing your views and perceptions about that system. Do you like the look of that system? What's good about it, what's bad about it? Does that process work for you? Is that output in the right format for where it's going on to? And there's a lot of information that you can actually*

gather you know by using those sorts of tools and techniques rather than just handing somebody a manual and saying there you go."

For commercial IS products, virtual participation through DREX may help customers understand the evolution of these products, including future plans and how particular features are being prioritised in the development planning process.

An important facet of participatory design is that it may help to create a shared accountability between the end user sponsors and the developers of an IS. Very often, IS departments and vendors are blamed for the adverse effects of a system implementation project. A more open, participatory design process, if properly managed, may help to reduce finger pointing or may lead to a situation in which stakeholders take a more careful interest in the product of their participation knowing that they will ultimately be held to account for the success or failure of the final product.

The DREX approach may facilitate discussion of 'lessons learned' between users and developers. More informed system feedback may be one result of allowing system stakeholders access to the actual design process that was followed as the system was developed. This entails a DREX mechanism that is a two-way channel, in other words, design rationale is communicated but at the same time new feedback can be incorporated into the design knowledge base.

"They may be able to give better information to the software developers in terms of you know, you don't understand this right? You know it's not really this way, it's this way. Your picture's all wrong. Whereas the customers that I've worked with they know their job, they don't really know the arguments that we go through in making design choices. I work with them more as a you know real world check, are we doing the right thing? But really after we've done all the design analysis and design choices."

Participatory design approaches must be balanced against the downside of design-by-committee discussed in the previous section.

Communication

IS development and implementation projects often entail a significant change to the work practices of the target user community. Resistance to such change has been identified as a major cause of IS project failure or lack of effectiveness. Participants in this study identified enhanced communication as a potentially key benefit of the

DREX approach in that communication was seen to reduce the threat of work practice change on the end user community.

> *" I would think that any product or service that comes out that asks the user to change you know their approach or their value system or the thinking, needs to be justified, you know. You know at least it'll go down easier if people understood the rationale for why you're asking them to do that, you know."*

Use of the DREX approach may also help to establish a shared understanding of the goals a IS project is meant to address, and how a particular design was formulated to address those goals. A constant interplay of requirements gathering and requirements verification as facilitated by a common, DREX-based design knowledge base could play a central role in this verification process.

> *"From what I've seen, right, and from what I've heard from my customers, what would make them the most happy is that they understand what the problem is and that we understand what the problem is. That we both understand what the workflow is, how to make the process the most efficient that we have considered all the technology options out there and we've chosen the one that solves the most important capabilities and first and leave out the less important capabilities."*

Communicating design alternatives, their selection criteria, and the arguments that directed a particular selection, for example, the results of user surveys or experiments that were used in feature selection, may help to rationalize these decisions in the eyes of the end user community. One type of example often given by participants involved responding to complaints about a given system feature with reference to the beneficial consequences obtained by the design in another, perhaps only distantly related feature.

The DREX approach may help users of an IS gain a better understanding of the business domain in which the system is used by showing how requirements and constraints from units of a dispersed organization were reconciled in the final design.

> *"So you know, if you're in sales or something, you're natural granularity is daily or weekly. Whereas if you're in the warehouse or parts or whatever, your natural granularity could be much longer out than that because your parts are coming, well a good example is if you're buying stuff from Korea, it's five weeks in the boat first of all for the stuff to get to wherever it is. So you've got to be ordering that much further out so sort of the window of where you're looking is much further out than where the sales guys are looking. And they often don't appreciate the kinds of lead times that there are and why, I saw this*

a lot, why forecasting needs to be much more precise than they suppose it does."

In the arena of commercial IS, exposing the design rationale of an IS may assist potential customers in their selection of competing systems as domain scenarios are outlined along with the design approach that was selected to meet the demands of the scenario.

Organisational Memory

One of the most significant benefits of design rationale capture is that it provides an organisational memory for both the designed artefact and the organisational 'stance' relative to the domain in which it is used. Organisational memories are valuable in both the short-term and the long-term in that they provide a central base of information about ongoing projects as well as a 'lessons learned' archive for both future IS, and organisational strategy.

"But I do feel that there's a lot more discussion that has to be repeated because there's not the up front analysis or there is documentation but nobody wants to read it so you just have to kind of keep reminding of what the decisions have been in the past."

One participant highlighted one of the key problems at the core of efforts to develop and build organisational memory systems and, more importantly, ensure that they are used and therefore able to achieve their potential value. This problem is reflected in the results described in this chapter, where the difficulties and apparent lack of interest inherent in actually capturing information prevents them from ever realising their value in use.

"I mean there's always something more critical. And yet there isn't because the truth is we're losing a lot of information this way, we're really losing it."

Credibility

A by-product of the enhanced communication facilitated by the DREX approach is increased credibility of the IS development organisation. Since a design rationale can show the constraints that bounded the design problem, the empirical evaluations that informed the design process, and how these results were woven into the design, end users can see that a given design was actually the product of a considered, professional approach rather than a series of ill-informed, ad hoc decisions. The

converse of this argument is that in cases where a design decision was the product of a less rationalised decision, for example selection of a development approach because of potential political consequences, DREX also exposes these faulty decision criteria.

> *"What kind of a person thought this was a good way to do it? What could they possibly have been thinking? Now for them to be offered an explanation that says well we considered you know, X, Y and Z and we went with Y for the following reason. As long as it's a half-way decent reason, which comes back to my basic model for explanation, is your reason a good reason for thinking that. As long as that's the case, so then you can say ah, it's a judgment call or as I like to say, men of good will could disagree about this, then I think you've really done something useful."*

> *"Like okay, I didn't get this feature that I wanted or I really wanted the system to be able to do this, it's not going to do this but I understand why. You know because it might be linked to the Y2K thing, the development takes too long or you know the people, the company that does, that does custom developments on this package is you know, they're not very good so, whatever it happens to be. But if they understand that then I think it helps in their acceptance of the system."*

Design Decisions

Exposing the details of an IS design can improve the design decision making process by reducing the effect of personal opinion and anecdotal evidence on the overall design. Exposing design questions helps deign teams to focus on critical success factors and to prioritise resource allocation. An additional benefit of sharing design information with end users is avoiding design-from-a-distance in the form of management fiat.

> *"If, if you're doing a system project and you've got you know, everybody who's working on it is at a management level, who's not hands-on, they can make all sorts of decisions that may be valid for whatever reasons they have, but if they don't understand the operational issues and how a system is actually going to be used, it can cause a lot of problems. So end users are very concerned with that. They want to know and understand and provide input as much as possible so that somebody else doesn't make a decision that is going to make their life miserable."*

Customisation

Exposing more of the design rationale to end users helps them to formulate more reasonable, informed customisation requests. Access to the design decision-making process helps them understand how design criteria, including especially design

constraints, are identified and applied to the selection of design alternatives in the customisation process.

> *"I can't think of a user who would sit in front of say SAP and you know for the last three months they've been doing something a certain way and suddenly why the hell, why can't I do it like this, this and this? And if they got the answer so what? It won't change anything probably or maybe they come up with a good idea. They've got the purpose of this is to not only satisfy their curiosity but is it to give an opportunity for an improvement to be suggested?"*

5.6 Discussion

This study provides a first empirical evaluation of the theoretical framework developed in Chapters 2 through 4. Recall that the framework attempts to do three things: provide a well-defined structure for IS explanatory content; propose design rationale as an approach to capturing this content; and, finally, situate the approach within the context of organisational information system development and use. This section discusses the extent to which the results of this study map to the theoretical framework, where the results call into question the viability of design rationale-based explanations, and where they suggest changes that might enhance theory fit by taking into account the pragmatics of the IS development and use context.

5.6.1 IS Explanation Content

Explanations of information systems may be classified using a multi-level set of categories. At the top level is the distinction between what are here termed *operational explanations*, distinguished by their immediate and direct utility, and *why* explanations, which provide *contextual* information meant to help the requestor understand the system within the rich array of factors that constitute its ecology, different subsets of which may be usefully assembled for any given scenario. Though not the focus of this research, operational explanations that answer questions related to *what* a system does, *how* these features are used, and *how* they work emerged as explanation concepts central to the intuitions of IS development professionals. However, answers to less immediate *why* questions are also important in that they help frame the context in which an IS was developed and used and therefore relate the structure of the system to the environment in which it was formed.

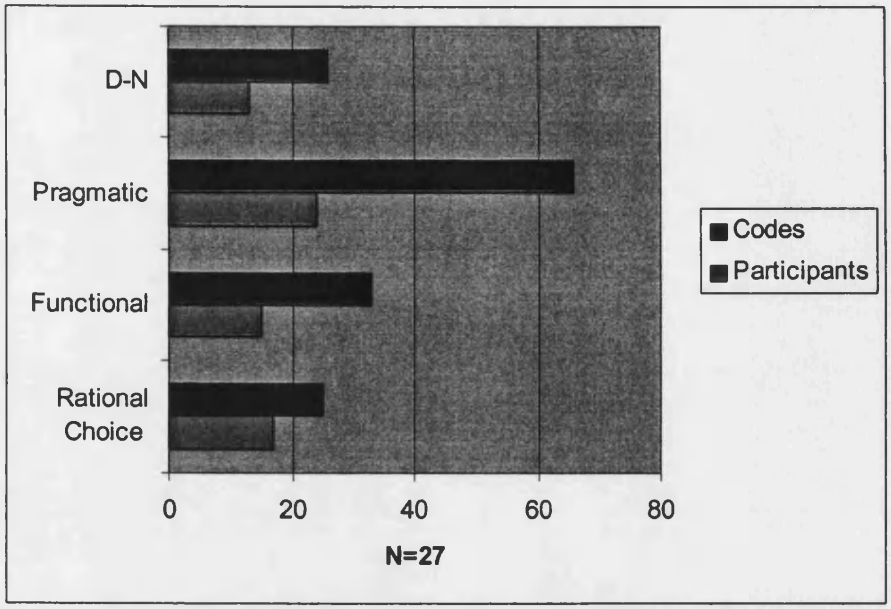


Figure 16 - Explanation Content Type Densities

The graph above reviews the theoretical densities of the different *why* explanation content types. As the figure shows, pragmatic explanations were the most dense, followed by functional explanations, rational choice explanations, and D-N explanations in descending order[1], though the latter three appear to have approximately the same level of importance. As can be seen, each of these information elements appears to play a role in IS explanation, with the most critical factor being the pragmatics of the information delivered, in other words, the relevance of a particular set of explanatory information to the scenario at hand.

One way to view the role of laws in IS explanations as emerged from this study may be viewed within the framework of the structuralist school of social theory. Structural explanations emerged from the work of Levi-Strauss, and provide accounts of social phenomena in terms of the system of rules and conventions, a set of behavioral constraints that comprise an accepted *order* and that determines how an individual operates within a social system (Manning & Cullum-Swan, 1994). Information systems development is bounded first by a set of constraints introduced by the domain, the system must function within the framework of laws that describe the domain, for example, financial accounting or meteorology. Design is further constrained by the technical aspects of system construction, hardware and networking

[1] No formal method for calculating a measure of theoretical density is suggested by the grounded theory approach. The ranking given here is obtained by simply summing the number of participants that discussed the concept and the number of times it was discussed (coded).

platforms, logical and physical data models, and the axioms of mathematics that constrain computation. Operating within these two sets of constraints, the evolving system architecture model is representative of a third set. This architecture model acts as a design scaffold that becomes increasingly restrictive as it evolves; prior design decisions necessarily constrain later ones in order that the design as a whole retains a level of coherence. The figure below depicts the role of various laws in the IS context.

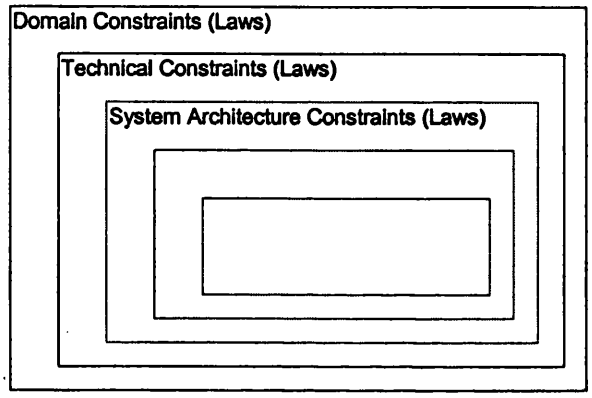


Figure 17 - Laws in IS Explanation

A coherent and communicable system architecture model may enhance the overall explainability of the system features and processes that are its constituents. For example, trainers at AT&T Bell Laboratories have recommended that before new developers are added to large software projects they be given instruction in the overall architecture of the system (Devanbu, et al., 1991). Conklin and Burgess-Yakemovic (1996), following Naur's (1985) idea of "programming as theory building", suggest that:

> *"the theory behind a system provides the conceptual background for the specific features and structures of the system design, and it provides a coherence and intelligibility to the huge number of otherwise isolated design elements."*

These theories may be conceived of as mental models of how a system is structured to perform a specific function (Norman, 1988). By making the system model explicit during the design process, DREX may provide a method for assisting system stakeholders to more quickly and accurately develop an appropriate mental model of the actual design model behind an IS.

The pragmatic theory suggests a process, rather than content oriented approach to providing explanations of an IS. Since the pragmatic theory suggests that explanatory

content is context dependent and highly interest relative, the prevalence of the pragmatic approach to explanation that was inferred from this study presents a significant challenge to attempts to describe a necessary and sufficient explanation content model. Given the difficulties inherent in the user modelling technology needed to support delivery of pragmatic explanations, an approach to dealing with the issue is to focus on the *design* of explanatory information. In other words, a focus on providing the information components necessary for the user to construct their own explanation relative their current information needs.

A key issue in explanation is deciding on what constitutes irrelevant detail, and what information is central to the explanation. In this study, some participants questioned the relevance of detailed technical information in an IS explanation. This result is resonant of reports from early expert system explanation research, where the intermingling of domain knowledge with the computer "artefacts" used to manage it was seen as a barrier to effective explanation generation (Swartout, 1983). An empirical question then is at what point does information become 'too technical' and whether, indeed, this question itself is contingent upon the context and the requestor's interest. Given the difficulties inherent in creating computer programs to determine context and human interest, this problem may be reframed as one of information taxonomy and information design. In other words, given that we cannot pre-determine what information constitutes a parsimonious explanation for any future scenario, the challenge is to provide information organised in such a way as to facilitate rapid, easy comprehension of the information taxonomy and rapid, easy access to information identified as relevant. The next chapter presents a prototype software system that attempts this design.

In the IS context, functional explanations may be provided with reference to either the micro-purpose or the macro-purpose of a given IS feature or process. A micro-purpose is an operational goal that shows how the feature or process performs some unit of work within the overall operation of the system (e.g., "This button saves the file"). The macro-purpose, on the other hand, shows how the feature, the process, or the IS as a whole helps to solve a domain problem (e.g., "This button saves the file for submission to the relevant tax authority"). An important third component, the relationship between these two purposes, may help to establish the relationship between the structure of an IS and the different dimensions of its purpose or strategy.

Participants identified this information as an important element of system explanation, one that is largely underrepresented in the documentation and help systems provided with most systems.

Identifying how an IS feature or process was designed to fit a particular purpose within the constraints of the IS development milieu may be one of the keys to useful why explanations of systems. IS development projects often take a long time to complete, and by the time the end product is in active use, project stakeholders may have lost sight of the original motivation behind a system, a motivation that may or may not be relevant to the organisation's current environment and priorities. This issue highlights the relation between the approach described in this dissertation and research into requirements traceability (Gotel & Finkelstein, 1994; Ramesh & Jarke, 2001), an approach that attempts to track the 'life' of a requirement through design, development, into production, and through subsequent modifications.

Though not as obviously relevant to users' interest as functional explanations, insight into how an IS design team and other stakeholders identify design questions, options, and criteria and weighed them relative to organisational priorities was identified as potentially useful in system explanations. By including the notion of human intention, rational choice theory helps to integrate and 'humanise' an otherwise potentially mechanistic worldview where function or purpose guides a design process constrained by physical laws and some of the law-like regularities found in the social world. Rational choice theory, especially as modified by Simon's idea of bounded rationality, does not entail a rational design rationale, but instead suggests a process in which human fallibility is a key determinant of the products of design. DREX may aid this process by promoting reflection on this fallible design process, and help improve the product of design by supporting a more participatory environment.

The study described in this chapter represents an examination of the micro-content of IS explanations, what Clancey (1986, p. 46) calls the "bottomless pit" of support explanations. The term micro-content is not meant to suggest a reductionist approach to the problem of defining explanation. In fact, quite the opposite is true. The micro-content of explanation refers to the facts that bind an IS both to its evolution in development and to its use context. If the strategic and structural components of a system are roughly analogous to its procedures or methods in the case of the former,

and its modular or class structure along with any underlying data model in the case of the latter, the micro-content is that which associates these abstract entities to the concrete ways in which the system is used.

5.6.2 Challenges to DREX

Several significant challenges to the DREX approach emerged from this study. Among the most significant was that many of the IS professionals interviewed felt that users have no interest in the details of the systems that they use beyond what is necessary to do their jobs. In particular, only information describing the operation of the user interface and specific task-related instructions are required to effectively use a system. While most of these objections were based on scepticism about the utility of extensive technical details in system explanations, others felt that there was no interest in any of the factors, technical, organisational, or otherwise, that drive the system design and development process. Specifically, some participants felt that there was no explanatory connection between the technical architecture underlying a given IS, and the actual use of that system. However, this result conflict with the idea that information systems increasingly represent both the form of modern work, and the perspective of the organisation towards that work (Bowker & Leigh-Star, 1994). An open question for later empirical study is exactly what kind of information IS end users find relevant, as well as how more technical information can be transformed to make it more useful to the end user community.

Another major issue that emerged was the effect of time pressures on the systems design and development lifecycle. IS organisations have come under increasing pressure to deliver their products more quickly, and this has resulted in a backlash against the use of detailed development methodologies that can radically increase the number of project deliverables and the time taken to complete them. Many IS professionals are questioning the benefits of following detailed, prescriptive development methodologies and formalisms. IS organisations are under pressure to deliver working systems faster, and the increased use of corporate benchmarking has resulted in a competitive framework where the contributions of IS development formalisms and methodologies have been called into question.

These factors suggest an environment in which any incremental effort added to the IS development process will be viewed with deep scepticism, if not open hostility. Some

participants felt that too much information provided to users would result in an increased support burden as they would be more likely to question certain features or become confused by the information they were given. Some also felt that opening more issues to debate would actually *decrease* consensus on system context and feature sets as the complexity of the IS design decision making process increased with significant numbers of additional participants. Clearly, the use of design rationale and design rationale-based explanations must be managed to support an inclusive development process while avoiding design anarchy.

Some participants felt that problems of IS usability may be overcome by applying sound user-interface design principles as an integral part of the development process. This argument suggests that when a system human interface is properly designed, only a minimal amount of additional instruction is required to make effective use of the system. This same argument may also apply to the form given to information that is provided to IS users. Some participants felt that the problem is not one of having the right information, but of having the right access or search mechanism to retrieve what already exists. Even where acknowledging that the information provided by design rationale might be useful to end users, concerns were expressed regarding the ability to convey this information in a format that users could easily comprehend. Some participants expressed the related view that users had access to the information that they need, but they are far more comfortable using support organisations, i.e., picking up the telephone, than they are with taking the time to search for the information they need. Making the quantity of information produced in an IS design more usable would involve both complex processing tasks and the development of an information taxonomy to support more intuitive information retrieval.

Making detailed design information available within the application might provide competitors with information crucial to their competitive position, and therefore might detract from their business advantage. Also, disclosing some kinds of information might jeopardise the security of the system or the confidentiality of information contained within. Clearly there is a need to couple any effort to expose a system's design rationale with parallel efforts to maintain the security of this information as well as the confidentiality of the designers and other stakeholders involved in the design process.

Another side of this privacy issue was expressed by some participants who were uncomfortable with the idea of exposing some of their less well-constructed software to public scrutiny. Not all designs are beautiful, but exposing these designs to at least internal public scrutiny may actually help reduce the number of 'bad' designs, or at the very least provide the contextual information that may help a critic to understand the factors that forced a design into a particular sub-optimal form.

Fischer (1999) argues for the development of tools to make users more self-sufficient with the software that they use. He describes as "domain designers" those that would become closely involved in tailoring software tools to the organisational tasks where they are used. The idea of involving end-user system champions in the early stages of a development project was discussed frequently in the interviews. These champions, sometimes referred to as power users or subject matter experts, act as intermediaries between the end-user community and the IS organisation. They are frequently given special training to enable them to provide support to other end-users and to further enable them to contribute to system development and implementation efforts.

5.6.3 The Promise of DREX

Despite the many challenges discussed in the last section, this study has also identified a number of ways that user access to explanations derived from an IS design rationale might contribute to usability and acceptance of a system. These advantages derive largely from an expanded sense of involvement in the systems development process, from the availability of an organisational memory that captures the institutional stance towards an area of work, and from enhanced communication about how a given system supports the mission of the organisation in which it is used.

Some research from the field of expert systems recommends against the primacy of experts in the knowledge engineering process and argues instead for the use of knowledge "clans" as the source of expert knowledge in a given domain (Stefik & Conway, 1982). These knowledge clans take the form of micro-communities of practitioners working on similar problems and employing similar "craft practices", typically within the same organisation. DREX may play a role in making explicit some of the tacit knowledge upon which these "knowledge clans" and "communities of practice" are based, and providing access to the full depth of the deliberations that resulted in construction of the tools they use.

Research into the nature of learning suggests that learning occurs when novice participants in a "community of practice" are given access to the intricacies of the practice in which they are engaged (Lave & Wenger, 1991; Seely Brown & Duguid, 2000). In the IS domain it is suggested that system users achieve deeper understanding of a system model when they are allowed to develop and evolve this model themselves (Seely Brown, 1986). Gruber and Russell (1996) refer to the explanations that people construct themselves from design rationale (as opposed to being inferred deductively by a program) as 'weak' explanations, but these are only weak in the computational sense, not in the potential explanatory power that accrues from them.

The development of a mental model is the development of an understanding of the artefact in question, whether it be a toaster, a tractor or a complex computerized information system (Rawson, 1987). Explanations of a system's behaviour help end users to form more accurate mental models of how the system works (Lamberti & Wallace, 1990). Clancey (1983) refers to this process as a 'forcing function' that, through an explicit representation of a design, facilitates the development of naïve users into more expert ones as they understand the justification for the system structure.

The results suggest that an explanation facility may act as a central store of information about an IS and as such facilitate communication among stakeholders in the system context. Such a central store might help end user communities reach consensus on system priorities when development issues and the arguments behind them are made explicit. Providing design rationale to the user community early in the system project may also assist with the critical task of expectation setting during new development projects. This aspect may be especially important for large and/or dispersed organisations, in that decision making knowledge can be shared broadly simply by providing access to the design rationale knowledge base. Such open access to this knowledge base may also support a more reflective, critical approach to systems development.

The more information that is made available to users of a new information system, the more likely they are to accept a new technology. By providing analysis and design information to end-users, they are less likely to perceive systems projects as being

inflicted upon them and more likely to use their new knowledge of system decision making to contribute to development and implementation efforts. An issue discussed by several participants was that organisations sometimes experience a time lag between when a system is ready to implement or market and when the necessary documentation is available. This lag often results in frustrating delays. DREX may help to close this gap since system information is gathered incrementally during the project lifecycle and, it is presumed, should not change once the system is finalised. The existence of a complete design rationale may provide a valuable aid to the documentation team when gathering information for more traditional printed manuals and on line help systems.

Information systems are considered to be one of the most important media for the management of organisational knowledge (Lerch, et al., 1997). This knowledge can take the form of the data or content managed by the system or, more important in the context of this thesis, in the inscriptions that are made upon a system design as it is shaped by the organisational context (Bowker & Leigh Star, 1994). Some research suggests that the use of tools by humans is characterised by situated, distributed cognition, where humans and tools interact in complex ways to perform complex tasks (Hutchins 1995; Latour & Woolgar, 1986; Spinuzzi, 1999). Systems and their supporting materials are sometimes described as existing in an 'ecology' of varying stability (Spinuzzi & Zachry, 2000), in which humans and tools interact relative to some domain. These ideas suggest that explanations of these tools and the ways that they are used are also distributed, for example, between the human interface to the tool, the documentation supporting the use of the tool, the minds of the tool users, procedure manuals that describe how the tool is embedded in a task domain, and other sources.

By exposing much of the tacit knowledge that is embedded in IS tools, DREX has the potential to support more knowledgeable and effective tools users. Access to a central store of system design knowledge that includes information about the context that affects the design may assist with the task of training new users of an IS, and this training would go beyond the what (features) and how (to use them) information commonly provided. The explicit nature of the information captured using the design rationale approach would help to expose much of the valuable *tacit* knowledge that underpins a system's design. A DREX knowledge base that can be accessed at any

time also supports just-in-time training, users can access information about the system design when they need it to answer a question or solve a problem.

According to Pearl (1996) "explanations are used exclusively for passing responsibilities. Indeed, for thousands of years explanations had no other function. Therefore, only Gods, people and animals could cause things to happen, not objects, events or physical processes." By providing a clearer picture of the process followed by system developers and laying bare the myriad of factors that impact the development process, not just technical but cost pressures, market pressures, management decisions, political issues, and others, end-users are more likely to develop increases empathy with the providers of the systems that they use. By showing the various factors that influence design the evaluation of design alternatives, end-users are given the opportunity to see the wide range of organisational, not just technical, factors that influence the course of systems development project. In this sense, explanations that expose the design rationale behind an IS have the potential to increase both the credibility of the design team and the confidence that users have in their systems. A design rationale provides clear information on the roles that various actors played in the development process and the decisions that they made. This provides a level of accountability that may promote a more thoughtful approach to the development process.

Elsbach & Elofson (2000) explored the relationship between the way a decision explanation was "packaged" and resulting perceptions of the explanation. Their background research suggests that decision rationales and their content are more important than the form of explanation delivery to perceptions of decision adequacy. They explored three factors associated with decision explanations: the use of understandable (i.e., simple) language in the explanation, the use of "legitimating labels" in the explanation that frame the decision process relative to professional and social norms, and the effect of decision explanation packaging "cues" in causing an evaluator to cease their evaluating and accept the explanation. They found that the most important factors were whether the explanation used simple language reflecting a desire to be understood and the existence of decision cues that caused them to prematurely (from a rational decision making perspective) accept an explanation.

Research into the basis of trust relationships between information systems and their users suggest that the more the source of a given message is perceived of as credible, the more effective the message conveyed (Lerch, et al., 1997; Mak & Lyytinen, 1997). In particular, clarifying the relationship between the functionality of a system and the rationale applied to the design of that functionality by the system designers is a key facilitator in the process of building trust between a system and its users. An information system can be humanised to some degree by showing how its designers struggled with a rich set of often competing criteria in the process of producing a system to meet the needs of end users. As information systems become ever more complex, explanations that can expose the justification for their behaviours will become more important to developing these trust relationships (Swartout, 1983).

This chapter began an investigation into the DREX theory by exploring the content of IS explanations, the challenges that emerge from attempting to capture this content as design rationale, and the potential benefits that may accrue if these challenges can be mitigated. There is a clear tension between the tangible costs of capture and the potential benefits of DREX, which are potentially more difficult to measure. The chapter that follows will explore this cost benefit equation, examine whether the information central to an IS explanation can be captured using the design rationale approach, and provide a demonstration of how this information can be made available to IS end users.

6 Study Two - Capturing & Delivering Explanatory Content

The purpose of the last chapter, Chapter 5, was to ground the theory developed in Chapters 2 through 4 in a base of empirical data collected from interviews with IS development professionals. While Chapter 5 focused primarily on *what* constitutes an IS explanation, the purpose of this chapter is to examine *whether* and *how* the explanatory content described in Chapter 5 can be captured from design deliberations, structured into a knowledge base, and then accessed from within the system that it explains. Recall once again the three research questions motivating the thesis:

> *Q1. What can philosophical theories of explanation contribute to the development of a framework for integrated explanation facilities?*
>
> *Q2. Can ideas from the fields of explanation systems and design rationale be integrated into the framework from Q1 to produce an implementable model for explanations of information systems?*
>
> *Q3. Is the model from Q2 operationally realistic, is it cost-effective, and can it be integrated into the IS development process?*

The purpose of the study described in this chapter relative to Q1 is to examine the kinds of information that are surfaced in the IS analysis and design process and compare and contrast these against the IS explanation framework developed in prior chapters. Relative to Q2, the goal is to explore the extent to which the analysis and design information, in the form of design rationale, may be mapped to the explanation framework, and to the issues that arise in attempting to build software to capture and deliver design rationale as system explanations. Relative to Q3, the study examines the viability of the approach in terms of its potential costs and benefits.

The methodology employed to investigate these issues was a case study IS design project. As part of this case study, two working software systems were developed. The *Drust* system is used to capture design rationale and to provide explanations. The *VentureQuery* system acted as the design subject for the case study; it is an application for building and publishing electronic questionnaires to the World Wide Web that uses a Drust knowledge base as the source of its explanatory content. These systems act as demonstration proofs-of-concept and are an attempt to operationalise

the thesis ideas by providing the tools needed to capture, structure, and deliver explanations derived from design rationale.

6.1 Empirical Software Engineering & Case Study Research

The study reported in this chapter is an example of a software engineering case study. Historically, software engineering as a discipline has been largely led by practice, not theory, with little knowledge transfer from research studies into the field, with the noted exception of human-computer interaction (HCI) research (Glass, 1996). Part of the problem may be that there is little agreement on the best way to study software engineering. To date, a wide range of methods including case studies, laboratory and field experiments, secondary analyses of industry data, and collections of anecdotes from practice have been employed to explore central research questions in the field (Jeffery & Votta, 1999). This lack of an accepted research tradition and of a mature, widely-accepted theoretical orientation has retarded the development of a common language with which researchers and practitioners can communicate and evaluate studies that attempt to integrate and progress prior work.

Another impediment to progress in the field is that software engineering projects are characterised by a high degree of uniqueness. A wide range of factors impact the form and progress of a software development project, of particular importance are the individual style and productivity differences among software engineers (Redmiles, 1993). This individualism means that software development is still inherently a craft industry, making the processes involved in production more difficult to study than, for example, the manufacturing processes for more tangible products (Basili, et. al., 1999).

Exacerbating these issues is the fact software engineering exists on the cusp of several different modes of science: the basic sciences such as physics that identify the properties of materials, the axioms from mathematics that underlie computer programs, the so-called special sciences such as psychology that describe how individuals behave, the social sciences, which are concerned with how groups interact in both the creation and use of technology, and finally the engineering disciplines, which attempt to integrate knowledge from all of these into increasingly high performance devices. Given this range of reference disciplines that inform the field,

clearly software engineering must foster a high degree of ecumenism in the research methods it employs to advance understanding. To this end, multi-mode, qualitative, and ethnographic methods are increasingly accepted as valid approaches to investigations of the complex phenomena that impact the IS development and use (Wixon, 1995; Seaman, 1999; Trauth & Jessup, 2000).

6.1.1 Case Study Research

One form of research that can be used to explore the process of software engineering is the case study method. Case studies range from true field studies of IS developers and implementers working in their organisational context (e.g., Cusumano & Selby, 1997; Newman & Sabherwal, 1996; Orlikowski, 1993) to more controlled (and convenient) studies carried out with the express purpose of exploration, theory building, and theory testing (e.g., Shum, 1991). Case study research is widely practiced, and its efficacy as widely debated, in the study of management information systems, information systems development, and software engineering (Murphy & Walker, 1999; Lee, 1989). This debate is just one component of a more general discussion around the appropriate balance of rigour and relevance in IS research (e.g., Benbasat & Zmud, 1999). As a still-emerging discipline, the field of IS has struggled to achieve academic credibility through continual refinement of its basic research methods while at the same time attempting to ensure that the problems addressed by these methods, and the results obtained, are meaningful in a practical context.

Despite its relative lack of rigour, case study research has many proponents in this debate; it is empirical, it considers research questions in context, and it plays an important role in helping to integrate theoretical development with IS practice. Case studies that are designed specifically to investigate a particular theoretical framework are known as *instrumental* case studies (Berg, 1998; Stake, 1998). The role of systems development projects as case studies that can be used to evaluate theoretical frameworks is increasingly recognised as valid (Redmiles, 1993). In this approach, the case acts as a sort of laboratory for the investigation of the theory and its constructs; McGrath (1995) calls studies of this type *experimental simulations*.

Lee's (1989) influential article describes a methodology for case study research that is both scientific, in the sense that it may be used to test existing theories, and exploratory. Lee's analysis of what he cites as an exemplar of scientific case study

research (Markus, 1983) focuses on that author's use of three theoretical frameworks and the predictions they entail for a particular case. By framing the observations gathered in this case relative to the three theories, and by comparing the predictions of these theories to the events that actually occurred in the case, Markus was able to show support for one of the three theories as well as evidence contradicting the other two. Among the criticisms of case studies such as this is that they are inherently subjective in their focus, and the generalisability of any finding derived from a single case is generally weak (Berg, 1998). However, Lee argues that generalisability from a single case study occurs when the study provides support for a theory, which can then be used as the basis to understand other cases. Of course, in a given case the theory may actually fail to account for events, and therefore be weakened or even considered falsified.

One argument for generalisability from single case studies is that by providing sufficient detail about the case, readers are able to infer from the relation of the case to the theoretical framework how the case might generalise to other settings (Stake, 1998). Yin (1984) calls this *analytic generalisation*, as opposed to the statistical generalisations supported by quantitative methods and argues further that single case studies, like single experiments, only support generalisations to the extent that they fit within a theoretical framework which is itself an aggregate product of a series of studies that form a *chain of evidence.*

Many studies have examined the system design process by analysing the process and product of design meetings (e.g., Moran & Carroll, 1996; Olson, et. al., 1996; Carroll, et. al., 1994; Kuwana & Herbsleb, 1993; Vliegen & Van Mal, 1990; Conklin & Begeman, 1988). Studying the communications of software designers and developers is recognised as one of the more effective means of analysing the nature and content of this activity (Seaman, 1998). The challenge in these types of studies is to balance internal and external validity (rigour) while at the same time ensuring that the study is defensible in terms of its ecological validity (relevance). Of course, efforts to achieve this balance take place within a framework of resource that bound the time and effort that can be applied towards completion of the research.

6.1.2 Software Prototypes

Two proof-of concept software prototypes were designed and constructed as part of this study. Software prototypes are commonly used as the basis for knowledge building in computer science, software engineering, and artificial intelligence research. Construction of working demonstration prototype systems, subsequent analysis of both the process and product of construction, and testing of the resulting tools in constrained use contexts is widely practiced in the design rationale literature (e.g., Conklin & Begeman, 1988; Lee, 1990; Ramesh & Sengupta, 1995). As yet, however, there has been little work on attempts to formalise prototype-building as a true and accepted research methodology. One conception on knowledge building in empirical software engineering is that it proceeds from "design a little" to "code a little" on to "test a little" with analysis at each stage (Pfleeger, 1999).

The epistemological status of the results gleaned from such proof-of-concept research is an open question, at least for software engineering research projects that do not take as their explicit goal the development of new logical or mathematical theorem proofs. What software demonstration prototypes do provide is an experience of the issues that arise in attempting to implement an idea in an executable program, issues that can be documented to inform, or offer challenges to, later research efforts. In some fields such as intelligent tutoring systems and artificial intelligence, design and development of working systems is one of the primary means by which knowledge has been progressed (Wenger, 1987; Dennett, 1990).

Clancey (1993b) has argued that the development of software artefacts is an example of applied research that represents not the production of new knowledge, but the transfer of basic knowledge to particular scenarios. However, this view is countered by several theorists in the engineering sciences (Pitt, 2000; Petroski, 1996; Vincenti, 1990), who hold that engineering knowledge exists not only as the use and evaluation of the product of pure or hard science, but as a independent body of knowledge with its own characteristics and qualities. Glass (1996) points out that many theories in the domain of software development have emerged from studying identified "best practices" in software development and that these theories evolve only through continuous attempts to apply theoretical concepts. Perhaps the best rubric provided for the 'build it' approach to research methodology is from March and Smith (1995), who

argue that tool construction is central to the study of information technology, but it is only useful if resulting reports include analyses of both *how* and *why* a particular architecture helps to improve practice.

6.2 Study Design & Procedure

This section describes the system design case that served as the basis of the study. The case involved analysis, design, and construction of a software system to create and automatically publish electronic questionnaires to the world-wide web. The system development team consisted of the researcher and a group of M.Sc. students engaged on the Analysis, Design, and Management of Information Systems (ADMIS) course at the London School of Economics; more information about the study participants is provided below. Project meetings began in November 1999 and continued through July 2000 with the core of original participants. Construction of the system specified by the design continued through November 2000.

A primary goal of the case study was to provide an IS design project of realistic complexity to act as a source for design rationale, and to capture and structure the design rationale in a system capable of providing it back to system users. An early and key issue in the development of the case was selection of an application domain for the project. The most important selection criterion for the application domain was that it achieve a balance between complexity and scope; a realistic level of complexity was required to ensure some level of explanatory depth and a limitation on the project scope was required to ensure that the project could be completed in a reasonable amount of time. The design team decided as a group that the target application would consist of a web-based question-answer system in the form of a venture capital-seeking 'game'. The central goal of the application was to help educate novice e-business entrepreneurs in the venture capital-seeking process. The role of the system would be to provide an electronic simulation of an interview with a venture capitalist. Initially, much of the analysis and design activity was directed towards determining the kind of information entrepreneurs might need to prepare for such an interview. However, based on the realisation that a general-purpose question-answer architecture had emerged from early design efforts, the purpose of the project eventually evolved from a domain-specific to a more general purpose electronic questionnaire builder and publisher. The venture capital domain continued as the reference domain for systems

analysis, but design decisions after this point also took into consideration the applicability and utility of system features for other questionnaire forms including applications in research and teaching.

6.2.1 Study Participants

The project began with 12 members drawn from the ADMIS M.Sc. course at the London School of Economics, with the researcher making up a team of 13. The team eventually dwindled to a core of seven participants as members either lost interest in the project or found they no longer had the time to participate. Remaining members maintained a high level of participation in project meetings and were generally diligent in completing their project deliverables, which consisted primarily of domain analyses. About half of the core project team had IS development experience and all had a strong interest in the process of IS design. The make-up of the team was interesting in its cultural diversity, members were drawn from eight different countries representing four different continents (UK, Spain, Greece, Turkey, USA, Yemen, India, Singapore, Malaysia). Once the first-cut design and working prototype were complete, several other individuals with interest in actually using the application to publish electronic questionnaires became involved in initial demonstrations and user trials.

6.2.2 Data Sources

Twenty-one meetings of the core design team were recorded in full on audiotape. An additional three meetings between members of the design team and various project reviewers and potential users were also recorded in full. Meetings averaged 90 minutes. All audio tapes were transcribed to full-text files resulting in over 400 pages of design meeting dialogue (156,352 words). With one exception, design meeting transcripts were very close to complete, in the one case a bad audiotape caused about half of the discussion to be lost. In addition to the design meeting tapes, other project artefacts that were analysed for their contribution to the design rationale capture included: domain analysis documents, design documents (e.g., flip chart drawings), various Unified Modelling Language (UML) documents, meeting agendas and notes, and e-mail messages between team members. In general, the design meeting

transcripts captured the core of the design rationale with these other materials primarily acting to support and summarise the results of these discussions.

6.2.3 Design Rationale Capture

A two-stage process was applied in the initial analysis of the design meeting transcripts. Transcripts were first loaded into Atlas/ti (see Chapter 5) and the dialogue of each meeting was coded to isolate those segments related directly to analysis and design, as opposed to discussions about project planning, for example, or the small talk that occurred in every meeting. In addition, a substantial amount of discussion related to purely domain topics as the design team worked to understand the venture capital-seeking domain that was driving the application design. Discussions of the domain that directly related to design, for example, the use of scenarios to perform "thought experiments' on the emerging design, were coded as design deliberation.

Coded text segments were fairly coarse-grained to ensure that the context of a given design deliberation was not stripped away. Based on word counts of the before and after transcripts, approximately 52% of the content of the design meetings related directly to design of the application. These coded segments were then loaded into a second Atlas/ti analytic unit, where dialogue was analysed and categorised using a code set designed to identify segments as contributions to the design rationale. The code set used at this stage appears in the table below.

Table 5 - Initial Design Rationale Code Set

Code	Description
Question	Dialogue identifying a design Question for discussion.
Option	Dialogue identifying an Option related to an identified Question, or sometimes suggesting the Question.
Criteria	Dialogue related to the Criteria being applied in the selection of an Option.
Metacriteria	Dialogue related to higher level design goals such as performance, flexibility, etc.
Argument	Dialogue representing the arguments for or against an identified Option, additional "backing" or evidence supporting the Criterion being applied.
Decision	Dialogue where a identified Option is being selected for inclusion in the design specification.
Scenario	Dialogue where use scenario is being used to either envision or evaluate a design option under discussion.
Class Diagram	Dialogue where a particular class structure (design artefact) is being discussed.

In parallel with this coding process, an initial, rough design rationale was created using the Drust software application designed and developed as part of this research. This tool is described in further detail later in this section.

QOC

The central tension in the design and use of representational media for the capture and translation of complex system requirements and design is between the need for representational *fluidity* in support of the design capture process, and representational *fidelity* to ensure that the captured design product maps adequately to the intentions of the designers. The Questions, Options, Criteria, (QOC), design rationale semi-formalism (Maclean, et. al., 1996) was selected as the representational medium for the study. This selection was made based on QOC's balance of ease of use and its representational fidelity. QOC is a relatively simple and sparse method for representing design rationale. This simplicity was deemed an essential trait in the context of this study as it was felt to most closely parallel the selection criteria likely to be applied in 'real world' project settings, where practitioners are unlikely to invest time learning a potentially more richly expressive, but necessarily more complex and difficult to use formalism. In this sense, the project was less a *normative* approach to the study of design, in other words one that attempts to prescribe how design should be performed, and more a *descriptive* approach that attempted to capture what actually happens in the design context (Klein & Methlic, 1990). This approach fit the motivation underlying the research as an attempt to analyse what explanatory content can be gleaned from an IS design project with minimal prescription for action.

Though QOC was used to represent design rationale, the Design Space Analysis (DSA) approach in which it is normally embedded was not applied in the study (see the Discussion section of this chapter for the effects of this decision). Design team members were given only a brief description of design rationale generally, and QOC in particular. QOC was used exclusively by the researcher to create a retrospective design rationale and these rationales were not used to inform and guide the system design process. One concern motivating this strategy was that forcing participants to learn and use QOC would have resulted in a lesser commitment to the project. Instead, some of the more mainstream IS analysis and design tools and techniques

including the Soft System Methodology, or SSM, (Checkland, 1981) and elements of the Unified Modelling Language, or UML, (Fowler & Scott, 2000) were employed as representational aids in and between design meetings. As discussed in Chapter 4, design rationale is typically employed to augment rather than supplant system modelling and specification formalisms and since SSM and UML formed an important part of the participants' M.Sc. course, their use was seen as both a promoter and a benefit of project participation.

A second break with the DSA model was that every attempt was made to include as much as possible from design meeting transcripts in the design rationale. The DSA/QOC approach is really designed to address only those design problems that suggest the need for a deliberate, reflective design process, in other words, only the most difficult of the design problems that emerge. The requirement that all design related discourse be transformed into QOC stemmed from the need for explanatory completeness in the design rationale. Though it may be the case that only those system components complex enough to warrant the DSA approach actually need explanations at this level of detail, the decision was made in this study to capture as much as possible in order to get the most complete view of the captured explanatory content relative to the framework developed in earlier chapters.

Various references served as guidelines in the construction of the QOC design rationale including training materials explicitly developed for this purpose (Shum, 1991, Appendix 11, Appendix 14). These references provided hints, tips, and other heuristics to help with using QOC. To the greatest extent possible, QOC was applied as documented by its developers and potentially useful extensions that were suggested in the context of use were generally avoided in order to maintain the simplicity of the notation. However, certain additional constructs were adopted to solve acute issues that arose in management of the design rationale data for purposes of the project reported here.

Drust

The Drust application was created to capture the design rationale produced in the case study project. Drust is a Java application that front-ends a relational database developed using the mySQL open-source DBMS (www.mySQL.com). Drust includes some of the characteristics of gIBIS in its focus on the manageability of large data

through the use of a DBMS and text-based forms (Conklin & Begeman, 1988). Drust does not include a graphical design space browser, a significant shortcoming but one that reflects the project emphasis on capture, storage, and retrieval of a significantly large design space. The following two sections describe the basic Drust design and the way in which Drust acts as an explanation server for its client applications.

A central design goal was to maintain to the greatest extent possible the parsimony of the QOC approach while judiciously including features to support DR-capture and explanation-server functionality. In particular, certain extensions were required to create and maintain the relationships between QOC argument structures and the design artefact under consideration. This requirement arose from the need to facilitate retrieval of a design rationale explanation based on context-sensitive or index-based explanation requests. A high-level view of the Drust architecture is shown in the following figure.

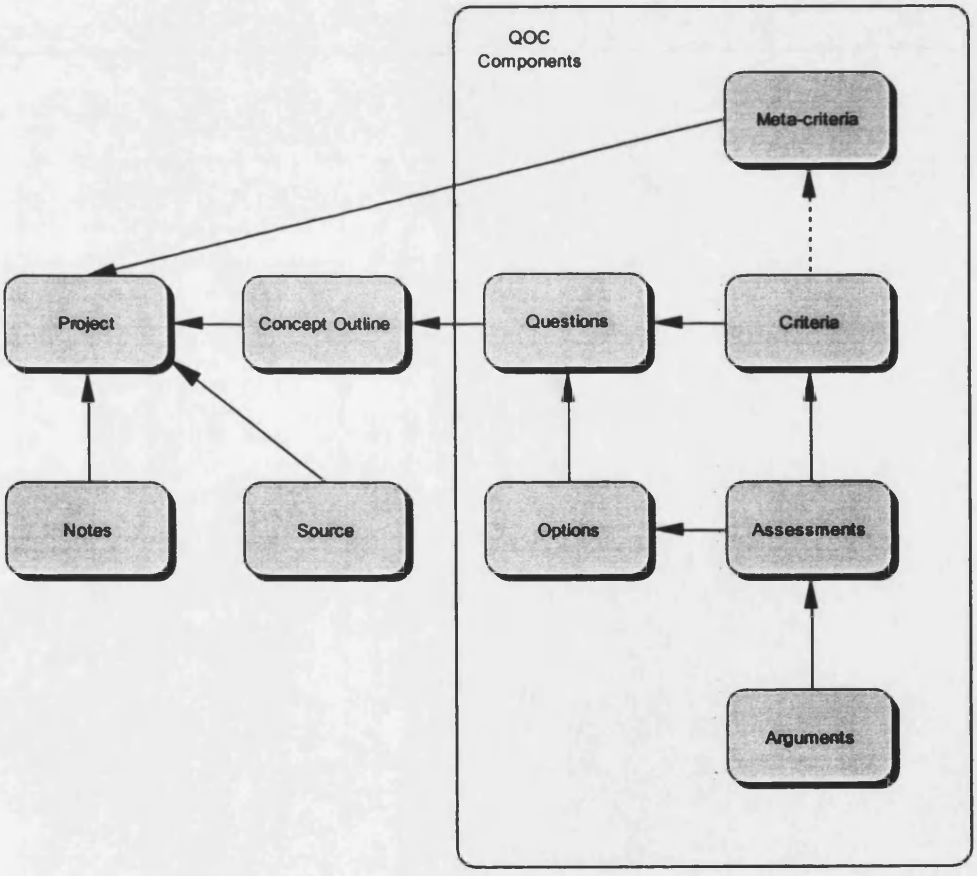


Figure 18 - Drust Architecture

In the figure above, arrow direction represents a relationship from a child entity to its parent. In Drust, all information related to a particular system is organised under a

parent Project. This allows a single instance of Drust to manage the design space for any number of on-going system projects. In addition to the core QOC entities, Drust also includes a Notes facility, which corresponds to the "escape" or "protonode" mechanism identified by Conklin & Begeman (1988), which is used to manage fragments of rationale that are not immediately categorised. The Concept Outline manages the relation between design target objects (e.g., a particular class or a particular window) and other, higher-level design targets (e.g., security), and the underlying design rationale in QOC form. This relationship is discussed in further detail below. The Source entity in Drust is used to capture the different contributors to the design rationale, for example, a design meeting transcript, a design document, or an email that produced an entry in the design rationale would be captured as a Source, and all of its contributions (e.g., Questions, Options, Criteria, etc.) would be tagged as originating from that source. Sources have been identified as useful data in other retrospective design rationale efforts (e.g., Shum, et. al., 1993) and were particularly useful in the process of finalising initial, rough QOC, when access to the original design deliberations was important.

The QOC-related components of the application support the basic set of entities and relationships identified in the semi-formalism (see Chapter 4). Both Options and Criteria have as their parent a Question, this was a pragmatic construct that eased construction of the user interface. Assessments relate Criteria to Options and may be elaborated by Arguments. Criteria may be instances of Metacriteria (such criteria are called *Bridging Criteria* by QOC's designers), though this relationship is not required. Finally, Questions may be derived from Options (the Consequent Questions of QOC) as a particular design issue is elaborated.

Drust employs a simple, text and form-based user interface for management of design rationale. In addition to being more economical to develop, this text based interface supports reasonably rapid creation and retrieval of design rationale information. Capturing the QOC data in tables supports more flexible retrieval and formatting of requested explanations, which will be discussed further in the next section. A graphical, icon-based, canvas-and-palette interface is planned for the near future and should improve creation and browsing of segments of the design rationale under focus, as well as support more natural design reasoning. However, this graphical representation will complement rather than replace the forms-based interface, which

is especially useful for the management of larger sets of design rationale data. The figure below shows the primary QOC window in Drust.

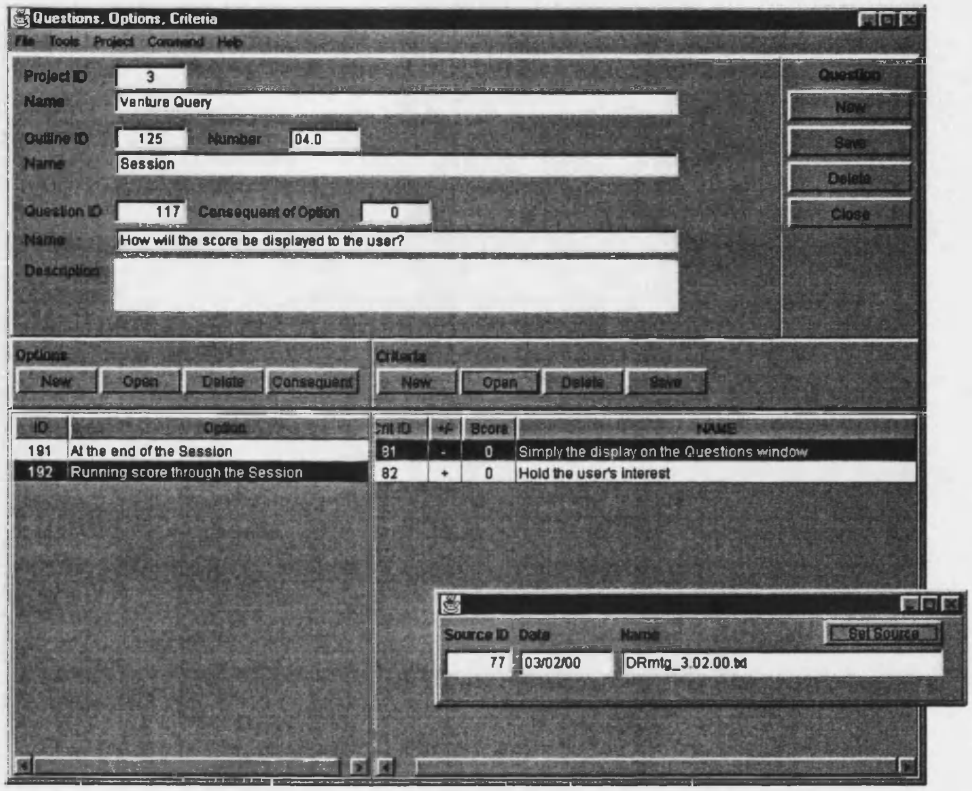


Figure 19 - Drust QOC and Source Windows

The following section describes how Drust was used to capture design rationale information for the VentureQuery project and how it acts as an explanation server for the *vqBuilder* application that was the product of this design effort.

6.2.4 Explanation Delivery

The software system developed as the subject of the case study reported here is VentureQuery, an application for designing electronic questionnaires, capturing questionnaire content, and publishing the questionnaires to the World Wide Web. VentureQuery design deliberations were captured as design rationale in Drust to create a knowledge base of explanatory content. The VentureQuery architecture consists of a Java application, vqBuilder, for the creation of questionnaires, and a set of Java servlets for publishing questionnaires to the web. vqBuilder is a reasonably complex application in that it embodies a particular model of how an online

questionnaire is constructed, how users navigate the questionnaire, the types of pre-defined and free-form answers that are supported, and how users' answers are captured and evaluated. In addition, the system includes functionality to support customisation of a questionnaire's graphic design and allows users to distribute elements of this graphic design (e.g., graphics files) to different host locations on the web. The system's complexity within a relatively limited scope of functionality made it a good test bed for the idea of design-rationale-as-explanation being investigated in this work.

vqBuilder implements the design rationale explanation feature provided by Drust. The next section provides a description of the vqBuilder application to provide the reader with a sense of the scope and complexity of the case study, and of the test bed application for which design rationale was captured. The section following describes how Drust acts as an explanation server for the vqBuilder application. To help clarify, the activity diagram below shows the relationship between VentureQuery, its component vqBuilder, and Drust.

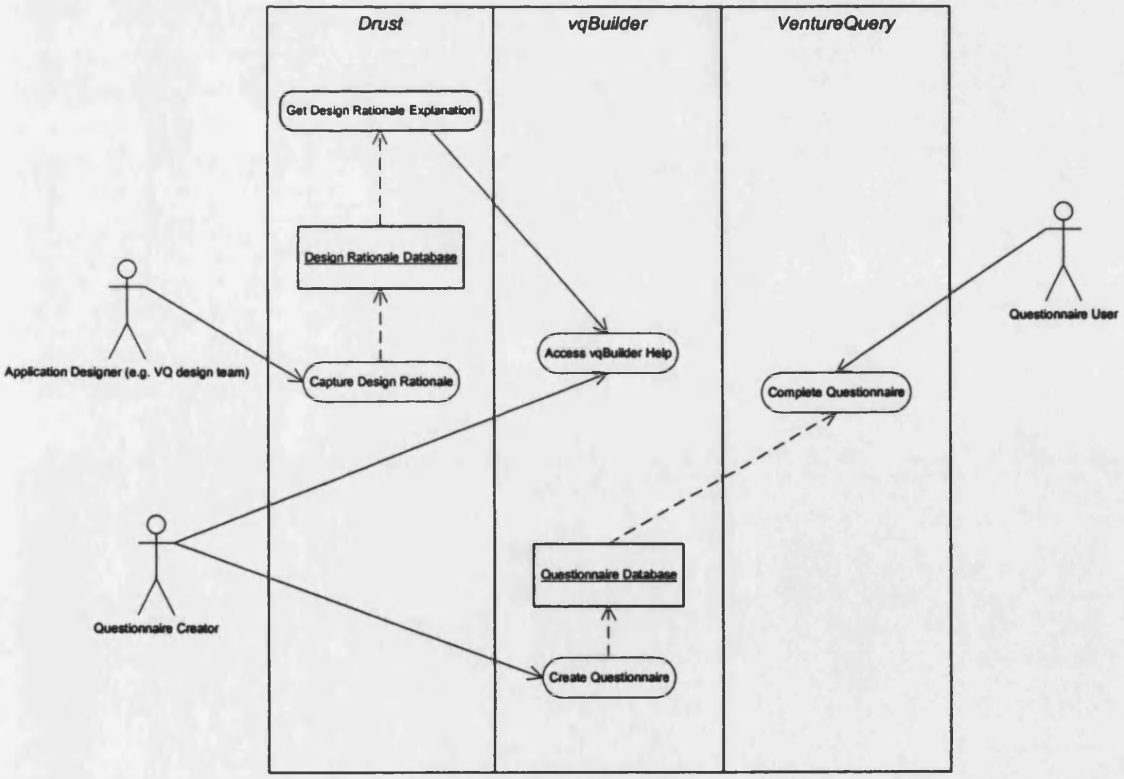


Figure 20 - Drust/VentureQuery Interoperation

vqBuilder

Based on a limited number of system demonstrations, vqBuilder is moderately difficult to understand and use. In order to successfully construct a questionnaire, first-time users need to first comprehend the system model metaphor that scaffolds the architecture, and then be helped to understand the functionality of specific system features. It is a forms-based Java application that embeds a particular model of the questionnaire design and creation process. The application metaphor is that of a board game, questionnaires are designed by envisaging a set of alternate *paths*, each alternate path consists of a series of board game *squares*. Each *square* (corresponding to a single web page in the web-based questionnaire) has one or more *questions* and associated *answers*. Answers for a given question have an *answer format*, which may be choices (checkbox or radio button user interface controls), or a text field which can also accept numbers as input. Users building questionnaires may also define *metrics* related to specific answers. Metrics have a dual purpose. First, they evaluate users' answers in cases where the user is provided with an analysis upon completion of the questionnaire, for example, if the questionnaire is an exam or quiz. Metrics are also responsible for managing changes to the questionnaire path based on prior user answers, for example, consider a questionnaire that has some common sections that all users are asked to complete, and others where different questions are asked depending on the gender of the user. The two sets of questions, one for males and one for females, correspond to different paths through the questionnaire. When the user answers the gender question, the metric for their answer, assume the user is a female, is responsible for deleting all male-specific questions from the questionnaire path.

In addition to the core questionnaire functionality, vqBuilder, also allows users to create an *Introduction* page, which may be used for a welcome message and/or for questionnaire instructions. Users building questionnaires may also choose to have the questionnaire analysed 'on the fly' and the results provided to the respondent upon completion of all questions. Users may also customise their questionnaire web pages with their own section headings, page headings, and graphics (e.g., a logo). Finally, respondents to a specific questionnaire may be randomised into a user defined number of study groups and provided with different questionnaire content depending on group membership. The figure below shows a representative vqBuilder window, in this case, where standard answer texts are created for a question with a radio button input type.

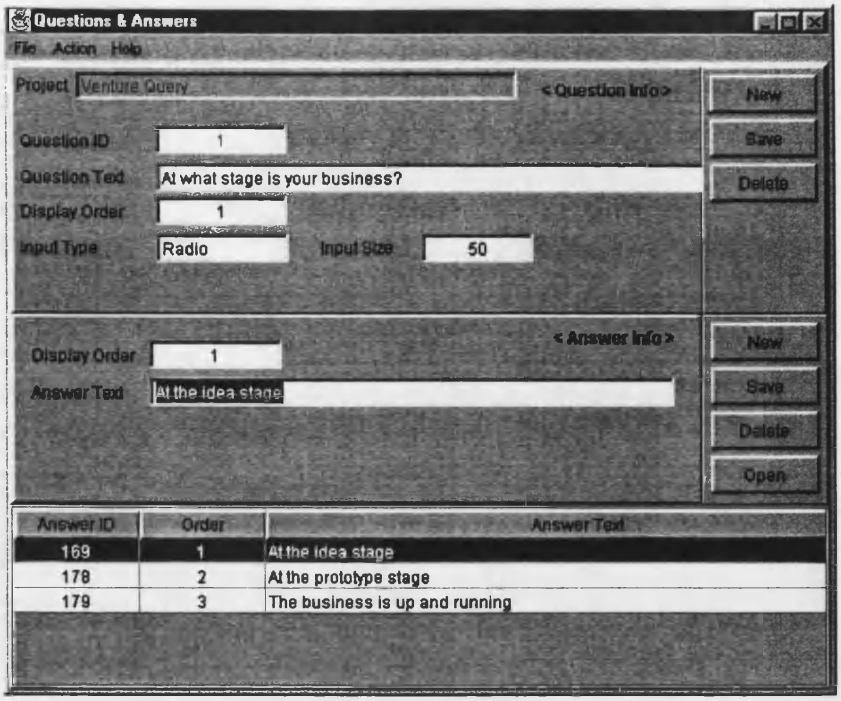


Figure 21 - Representative vqBuilder **Window**

vqBuilder stores the structure and content of a questionnaire in a database which is accessed by VentureQuery when a respondent visits the home or start web page defined by the creator of the questionnaire. This home or start page is the only static component of a questionnaire, all other pages are generated dynamically by VentureQuery as the respondent moves through the questionnaire path. Respondent answers are stored back to the database and may be exported to other applications such as a spreadsheet or statistics software package.

Drust Explanations

The QOC-based design rationale for the vqBuilder application, captured and managed by Drust, represents the knowledge base used to provide explanations. Drust acts as an *explanation server* for any client application that has had its design rationale captured. Explanations are provided based on the design rationale captured by Drust using a set of extensions to the *JavaHelp* architecture provided by Sun as an extension to the core Java Software Development Kit (java.sun.com). The only client-side programming required is inclusion of a Help menu item and an event handler for this menu item that calls JavaHelp. The JavaHelp architecture consists of a Java class library and supporting XML and HTML files and is designed to facilitate adding online help functionality to virtually any software application, much like Microsoft's HTML Help (msdn.microsoft.com) or eHelp's RoboHelp (www.ehelp.com) that are

used to provide standard help in many commercial software applications. JavaHelp presents help information using the tabbed dialogue window familiar to users of PC and Macintosh software applications, The window includes tabs for help Contents, Index, and, optionally, Search; another optional pane for sub-topics, and a pane to display the content for the currently selected help topic. The figure below shows an example JavaHelp Navigator window.

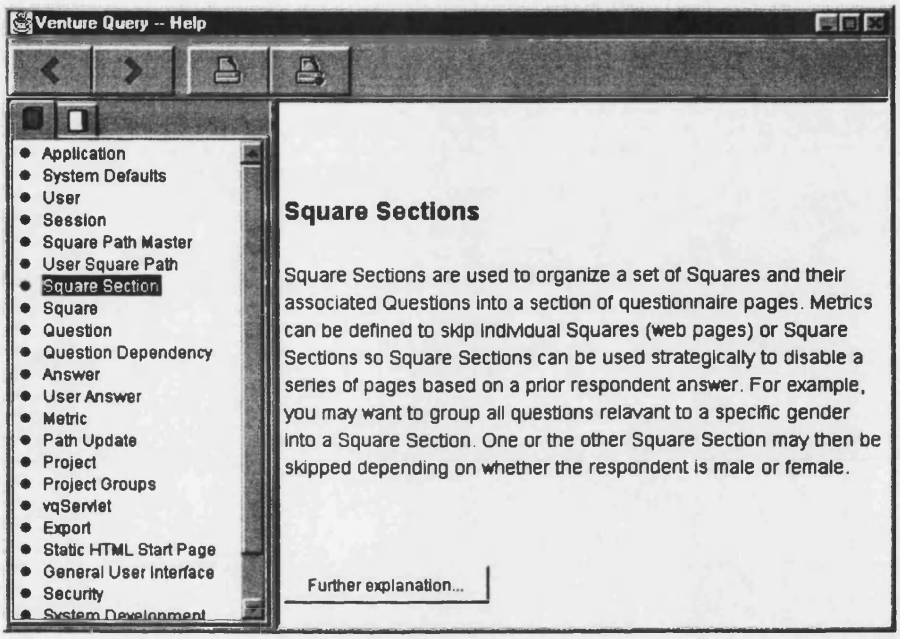


Figure 22 - JavaHelp Navigator Window

Standard JavaHelp uses static XML files to structure and present the table of contents and index of a set of content files, also static, that are in HTML format. The standard JavaHelp architecture is shown in the figure below.

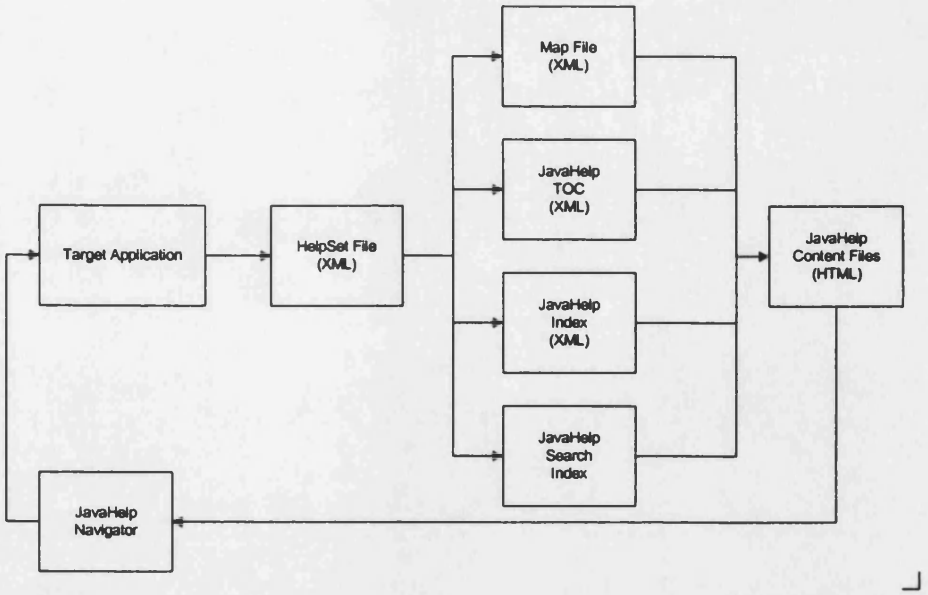


Figure 23 - JavaHelp Architecture

In standard JavaHelp, a target application for which help is being provided has an associated HelpSet file that points to a Map file, a TOC file, an Index file, and an optional Search Index. The Map file is used to associate application objects and other concepts, such as a window or button, or a feature such as security, to the HTML file containing the help content for the object. The TOC and Index files provide the structure and content for the JavaHelp navigator TOC and Index tabs and associate the labels that appear in the JavaHelp Navigator to the help concept in the Map file. For example, a label such as *Managing Sessions* is listed in the JavaHelp Navigator's TOC and Index tabs. The TOC.xml and Index.xml files relate the label *Managing Sessions* to a help concept called *sessions*. The Map.xml file then associates the concept *sessions* to an HTML content file called sessions.html. When the user requests help on the label Managing Sessions, JavaHelp uses this chain of associations to locate the sessions.html file and return this content to the JavaHelp Navigator.

The Drust explanation architecture extends JavaHelp to support delivery of dynamic help content and design rationale-based explanations from a Drust project. Instead of relying on the static XML structure and static HTML content files, Drust uses a Java servlet (a server-side, semi-autonomous application) to respond to help and explanation requests with content that is extracted from the Drust database, formatted into HTML dynamically, and then delivered to the JavaHelp Help Navigator. Drust is also responsible for creating the JavaHelp HelpSet, Map, Table of Contents, and

Index XML files. These are derived from the Drust concept outline and can be generated any time the concept outline changes. Since Drust is a dynamic explanation server, this means that help content can be created, appended, or amended at any time and immediately made available to the target application. The figure below shows how the Drust explanation servlet works with the JavaHelp system.

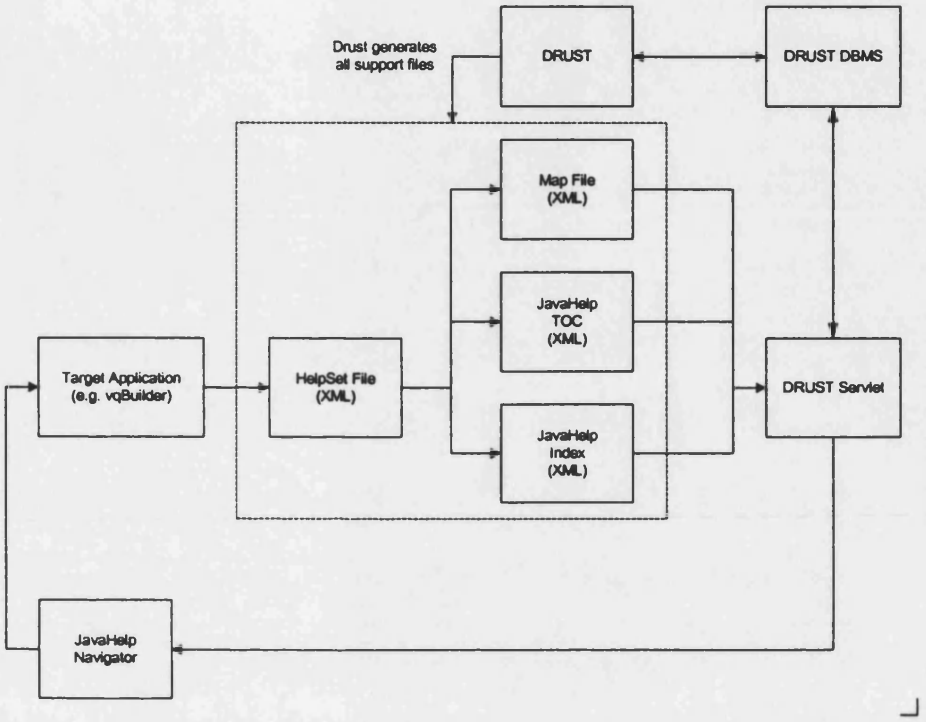


Figure 24 - Drust **Explanation Servlet & JavaHelp**

A target application such as vqBuilder that implements the Drust help facility is able to provide two levels of help content to a user. When the help menu item is selected in the target application, users are presented with a list of help concepts corresponding to the project outline that has been created for the application in Drust. When a help concept is selected, the first set of information to be retrieved is the standard help that has been created for the concept (shown above in Figure 21). Standard help provides a brief overview of the concept and how any related features are used in the target application. From the standard help page in the JavaHelp navigator, users may request additional information be pressing the *Further Explanation...* button. The Drust explanation servlet then provides additional content based on the QOC that has been captured for the concept. The figure below shows the initial level of QOC that is provided to the user.

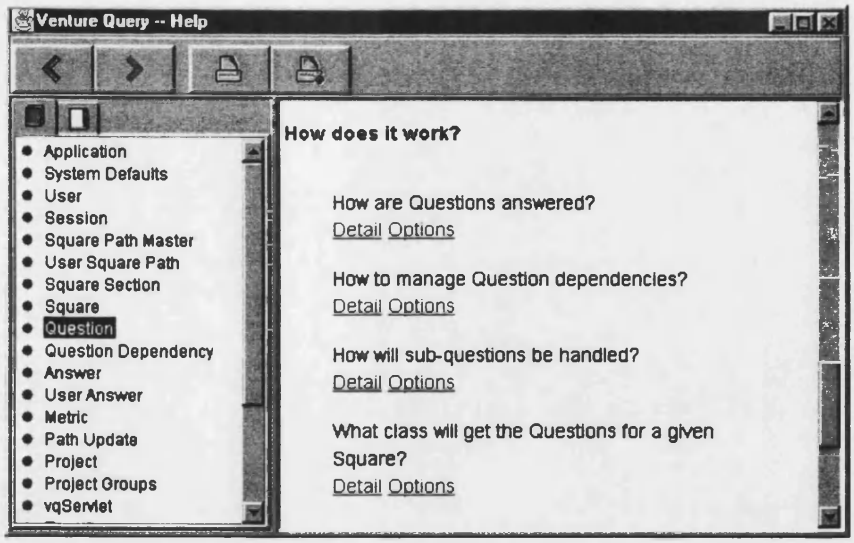


Figure 25 - Drust QOC Explanations

The QOC Questions captured in Drust are categorised across two dimensions. First by the help concept, which generally relate to the Java class that implements the feature, and second by which of three questions the QOC Question is deemed to address: *What is it?, How do I use it?,* or *How does it work?* The use of these three questions as high-level classifiers was derived from the results of Study One reported in the last chapter. More information on the relation between these three questions and the QOC Questions captured as design rationale is reported in the next section.

Once presented with the QOC Questions related to a help concept, users may choose to review the Options that were considered to address the Question and the Criteria that were applied in the selection or rejection of an Option. The figure below shows a Question that has been expanded to display its constituent Options, and the Criteria that were applied in the choosing between them.

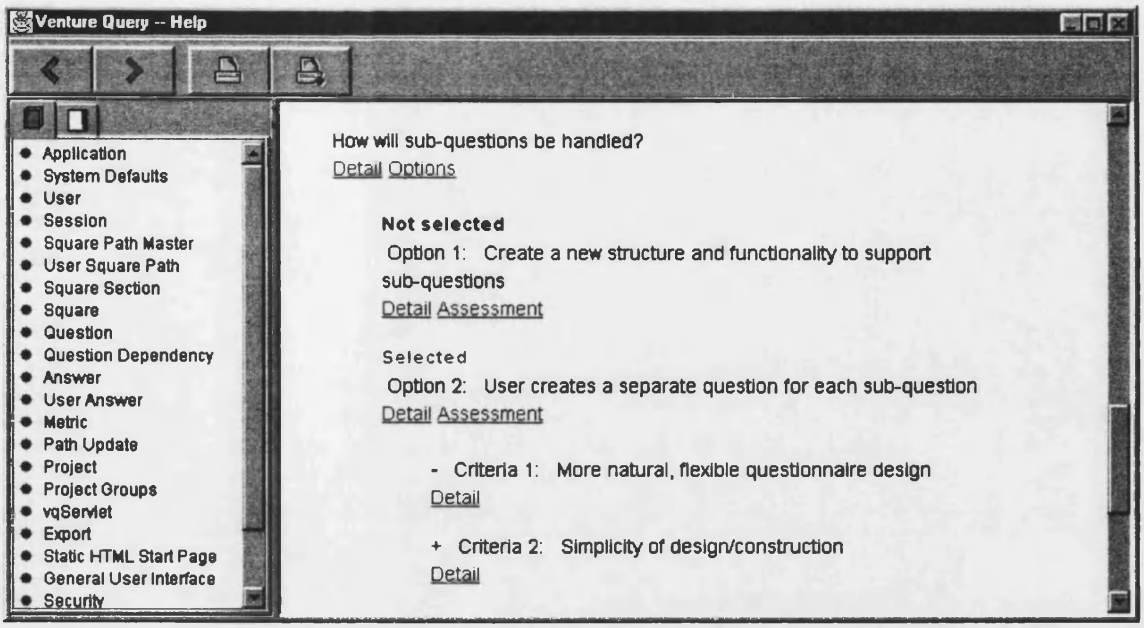


Figure 26 - An QOC Structure displayed by Drust

The section that follows examines the extent to which the Drust explanation delivery model is able to provide explanations of a target application's features and functionality using QOC-based content captured during the process of application design.

6.3 Explaining with Design Rationale: Results

The sections that follow report the results of the VentureQuery case study. Recall once again the three research questions underlying the thesis:

> *Q1. What can philosophical theories of explanation contribute to the development of a framework for integrated explanation facilities?*
>
> *Q2. Can ideas from the fields of explanation systems and design rationale be integrated into the framework from Q1 to produce an implementable model for explanations of information systems?*
>
> *Q3. Is the model from Q2 operationally realistic, is it cost-effective, and can it be integrated into the IS development process?*

The next section, 6.3.1, focuses on the information captured as QOC design rationale and provides an analysis of this information in relation to the explanation theoretical framework developed in previous chapters. Section 6.3.2 explores the issues that arose in the implementation of a software system to deliver this design rationale content as embedded explanations. Finally, section 6.3.3 discusses the potential costs

and benefits of this approach to extending IS online help with design rationale explanations.

6.3.1 Capturing QOC as Explanatory Content

This section provides an overview of the QOC that was captured in the *VetureQuery* case study. Though the usability of QOC was not the focus of this research, the section also describes some of the issues that arose in transforming 'raw' transcripts of design discussions into semi-formal design rationale. Counts of the different elements that were captured along with some basic, descriptive statistics are provided along with an analysis of the content of the QOC in relation to the explanation theoretical framework. The following table shows the kinds and number of different QOC elements that were captured.

Table 6 - QOC Element Counts

QOC Element	Number Captured
QOC Outline Elements	25
Total Questions	151
Total Options	339
Total Criteria	122
Meta-criteria High level design criteria that seemed to pervade discussions and which gave rise to specific (bridging) Criteria being applied to Options.	21
Bridging Criteria Specific instances of a Meta-criteria applied to the resolution of a single design Question.	87
Consequent Questions New Questions that arose as a result of a selected Option.	32
Assessed Option-Criterion Pairs Option-Criterion pairs for which an explicit assessment, + or -, was derivable directly from the meeting transcripts and other materials.	114
Un-assessed Option-Criterion Pairs Option-Criterion pairs for which no explicit assessment, + or -, was derivable directly from the meeting transcripts and other materials.	322

The QOC Outline

A problem encountered early in the project was management of the mass of data available when DR construction began. To help with QOC organisation and retrieval, a system of high-level categories was created in the form of an outline and used to group related QOC structures by referencing an outline element from the QOC Questions. Several different categorisation schemes were attempted before one based largely on the emergent class hierarchy (object-oriented structures corresponding roughly to domain entities) was settled upon. Since an object-oriented analysis and design (OOAD) approach was the explicit but loosely followed development methodology, the process of class, class attribute, and class behaviour discovery presented a fairly natural order to design deliberations and to the design targets or artefacts (the Java classes) being designed. These design targets were arranged in a shallow hierarchy with the application at the root level and concrete classes at the lowest level. QOC structures that emerged from meeting transcripts were therefore categorised according to their related design target. This explicit link between the design artefact and related QOC structures also proved useful in the process of explanation retrieval as described in the next section.

In cases where the design target was as yet unclear, Questions were categorised as unstructured, or a utility category, such as System Administration and System Testing, was created to serve as its parent. QOC structures related to the specific attributes and behaviours of classes were captured under the parent class. The figure below shows a fragment of the outline used to organise QOC structures for the VentureQuery test-bed application.

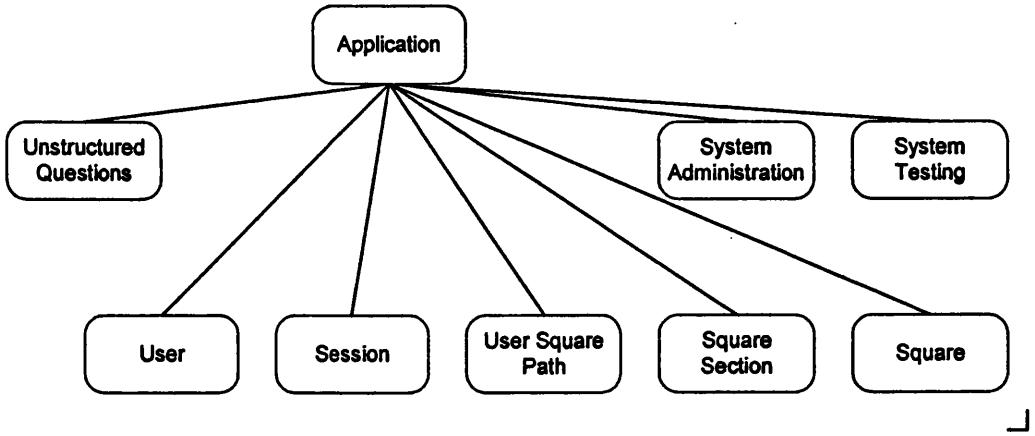


Figure 27 - QOC Outline Fragment for the VentureQuery Project

Retrieval of QOC structures for modification and extension was one of the more problematic issues that arose in capturing of the design rationale and a key benefit of the addition of the Outline grouping mechanism was in QOC retrieval. Once the number of structures rose to over 100, it became increasingly difficult to identify whether a new deliberation related to a pre-existing structure or not, and even in cases where it was clear that there was already a related structure, identifying the Outline category under which it could be found was often difficult. The approach adopted was to create redundant QOC structures during analysis of design transcripts and then to meld these during the process of transforming this rough QOC into finished structures.

The QOC outline also served as the basis for the standard help table of contents provided by Drust. Each outline topic relates to a brief text describing the element and its functionality. As discussed in Chapter 2, the minimalist approach to system documentation suggests limiting the amount of text incorporated into the documentation set to short, and goal-oriented passages related to topic headers that as clearly as possible reflect the structure of the system (van der Meij, 1992). This objective is accomplished since the QOC Outline maps the application class hierarchy to standard help content and then further to the related QOC structures.

That QOC with the Outline extension relates design rationale to the attributes and behaviours of an individual class is a potential strength in the object-oriented analysis and design process, which is predicated on encapsulation of a single object's elements with only the interface exposed to other objects. However, at times both designers and users need to consider the process interactions of several objects in order to fully understand the functionality of the system as a whole. QOC was less adept at displaying these interactions and dependencies using the simple, hierarchical categorisation mechanism employed here and it seems likely that to address this problem would require either a more elaborate, networked categorisation mechanism, making capture of the QOC more difficult and time consuming, or the development of an intelligent explanation retrieval tool able to traverse the QOC structures, identify elements related to an explanation-seeking question, and present the resulting explanation content coherently.

Questions

A total of 151 QOC Questions were captured from the design meeting transcripts and other project materials. This number seems lower than should be expected from a design project of this scope and scale, but there are a number of factors that may help to explain why this is the case. The first and most important factor is that the DSA approach, which explicitly frames the design process as a series of questions with corresponding options and criteria, was not followed. Though a design question is the natural entry point for creating a new QOC structure, the design process does not appear to naturally and consistently flow from question to question in series. Design deliberations did sometimes start explicitly with a question easily formulated as QOC, for example, "*What navigation aids will be provided to help users work through the system?*", and in other cases design options emerged which suggested the question, for example, "*Let's add a page menu to the system's home page.*"

However, in many cases the design question related to a given deliberation was far more difficult to derive. As has been pointed out with the use of QOC, one of the challenges with transforming design discourse into semi-structured QOC is this process of identifying and naming design questions, as well as capturing the sometimes tenuous relationships between different questions and between design deliberations and previously identified questions (Bellotti, MacLean, & Moran, 1991).

Full-text transcripts of the design meetings presented an almost overwhelming amount of detail to be ordered into QOC. Discussions moved from high-level questions to ones that dealt with the micro-structure of the application being designed. This serendipity does not necessarily reflect a unique lack of structure, software design is characterised as proceeding not in an exclusively top-down or bottom-up process, but as an iterative process, which includes both top-down and bottom-up thinking at all different stages of the design as well as the generation of hypotheses and goals that might redirect project team efforts and potentially change course of the entire project itself (Carroll, 1998). Indeed, this lack of process prescription was an explicit goal of the research design.

Another issue impacting the nature of the design deliberations was that a substantial amount of discussion in design meetings related to 'pure' domain concepts, in other words, these discussions were representative of the design team's efforts to explore

and understand the project domain of venture capital seeking, considered by the team as a prerequisite to design an application capable of simulating venture capitalist interviews. It may be that these domain-related discussions make an important but implicit contribution to the formation of the team's shared design model, given the amount of seemingly necessary information that appears to be missing from more explicit design discourse.

Another possible reason for the fairly small number of Questions may be the disproportionate amount of time spent exploring, discussing, and debating design approaches to a few significant and difficult design issues, such as how to manage the dynamic *square path* that leads respondents through a questionnaire session, how to manage the dependencies between different elements of a questionnaire, and the best approach to calculating a respondent's score on a given questionnaire. These are also cases where having a more complete, DSA compliant QOC that was updated after each design meeting may have proven useful, as much of the time spent on these issues included reviewing prior deliberations related to these issues at the start of a new meeting.

To better understand the nature of the Questions that were captured, an attempt was made to classify each question according to the *operational explanation* framework identified in Chapter 5, as design questions seem most closely related to these rather than the *why?* explanations provided by option-criterion pairs. The following table shows the results of the classification.

Table 7 - Explanation-seeking Questions and QOC

Operational Explanation	QOC Questions Count/Percentage Example
What is it?	56/37% What is the purpose of User Answer? What are the attributes of User Answer?
How do I use it?	35/23% What User Answer input formats are supported? Can Question wording be varied?
How does it work?	54/36% How are User Answers validated? How are Question dependencies managed?

Operational Explanation	QOC Questions Count/Percentage Example
Other	6/4% Who will system test the application? Who will own the rights to the application?

Object-oriented analysis and design proceeds by identifying candidate classes along with their candidate attributes and behaviours. Communicating the components of a class along with the reasons why certain attributes and behaviours are encapsulated in the class is important for system maintenance and dependency management, as well as for communicating to users what is available in terms of data and functionality. However, these sorts of structural questions are not particularly amenable to the QOC format. As is seen in some of the examples above, these questions often suggest an enumeration, rather the a set of distinct, separate Options. For example, questions such as *"what are the elements of the system model metaphor?"* or *"what are the attributes of the class User Answer?"* Structural decomposition such as this is one of the central processes in complex software design (Jeffries, et. al., 1981) and a recurrent issue with the use of QOC was the handling of design questions that were structural in nature. Some design questions have a relatively large number of candidate options, any *combination* of which may be selected for implementation. Obviously, enumeration of each of the different combinations of elements that might be created to answer an enumeration question such as these is impractical, and use of Options to list potential components goes against two tenets of QOC in the DSA model: that each option be a discrete solution to a well-formed design Question and that each Criterion relate to each of the Options.

However, in the approach adopted here, these structural Questions were captured as QOC in Drust in order to maximise the completeness of the design record and to centralise to the greatest possible extent the design information design information in QOC format available for explanations. In any event the QOC format proved quite amenable to capturing a list of candidate options in response to these structural and enumeration question. Candidate system components that had an explicit argument for being included have related criteria that capture those arguments, as would those components dropped from inclusion in the design. However, these types of Questions

also gave rise to many of the gaps in the QOC where Options either had no Criteria generated to help select from among them, or were not considered relative to those criteria that were generated. These issues are explored further in the sections of Options and Criteria that follow.

The QOC/DSA approach includes as one of its relationship types the idea of questions that emerge as a consequence of a selected design Option, in QOC these are *Consequent Questions* (MacLean, et.al, 1991). A total of 32 Consequent Questions were derived from the design sources. Relative to the issue of structural or enumeration questions identified above, this number may have been much higher if class hierarchy or composition questions had been treated as existing in a chain of successively more detailed questions emerging from consideration of a high level question such as *"what are the attributes and operations of the Square Path Master class?"* In a similar vein, design questions were found to often suggest immediate additional questions, without reference to a particular, previously selected Option. However, this approach would exacerbate the QOC retrieval and relationship management issues discussed earlier, as well as make further complex the process of navigating the explanations produced from such a deep network of inter-related questions.

Options

As the summary data reported in Table 8 above suggest, the mean number of Options identified for each Question was 2.25. However, the standard deviation of Options per Question of 2.24, and the frequency chart below, both show how much the number of Options per Question actually varied, including how often only one, or even no Options were identified in response to a Question raised in the design.

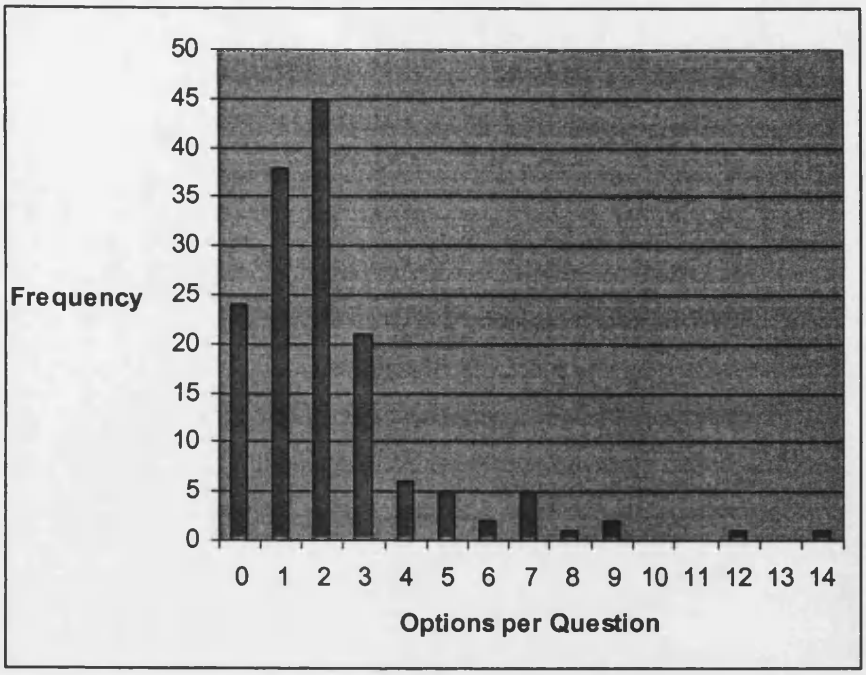


Figure 28 - Options per Question

Clearly this dearth of identified Options for design Questions which had been explicitly posed raises some interesting questions about the design process itself, and about the potential value of applying the DSA approach to force multiple Options to the surface for every design Question. As discussed in Chapter 4, design options not chosen for implementation play a potentially important role as explanatory *contrast* classes. They help to explain a design by showing how it may otherwise have been and by showing how design criteria were applied in the selection of the 'winning' Option.

One interesting observation is that meetings with potential users of the application tended to produce more Options relative to the amount of time spent in these meetings than those meetings that involved only the design team. One possible explanation for this is that potential users, without a significant time investment in realisation of the Options they generate, are not bound by the resource constraints that so closely guide the actual design team (more on these constraints in the Criteria section next). In other words since potential users of an application have little to lose and much to potentially gain in terms of both the number and complexity of the system features they envision.

Design questions sometimes took the form of yes/no questions, as in *"Allow respondents to skip answers?"* These correspond to the so-called *null position* in design rationale, an Options that essentially mean "do nothing" or "this doesn't make

any sense" (Conklin & Begeman, 1988). These cases often helped to tease out the identified requirement or use scenario, captured as criteria, that force the question into the design space. This information is essential in helping to relate a design artefact to the *functional explanation*, or the purpose, for which a given system component is being considered.

Criteria

Arguably the most important explanatory element of the QOC representation are the Criteria that are identified by the design team as salient, and the manner in which they are applied in the selection of different design Options. As shown in Table 8, a disappointing total number of Criteria, 122, were identified in the design meeting transcripts and other supporting materials. In many cases design Options were considered without explicit criteria being applied in the deliberation, or the Criteria were sometimes very subtle and difficult to derive from the deliberative context.

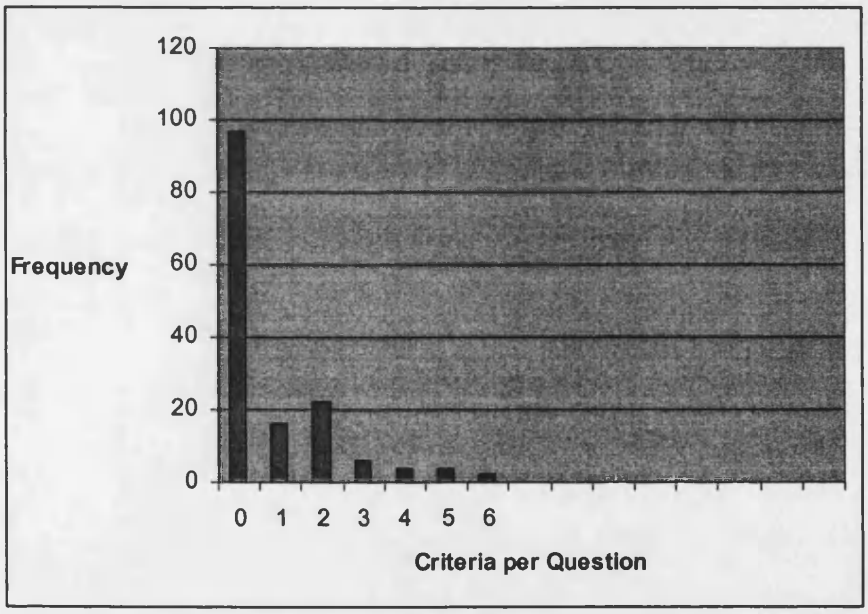


Figure 29 - Criteria per Question Frequencies

One of the most striking results shown in the figure above is that 98 of the 151 design questions identified, 65%, were considered with no explicit criteria applied in their assessment. Of these 98, 43 questions had more than one option identified to address the question, so the lack of criteria to assist with selection of an option was even more surprising. Even in cases where multiple options were identified for a question, and multiple criteria were identified to help with option selection, the option-criteria

matrix of the QOC structure was often only sparsely populated, in other words, not all options were considered in light of all criteria, as prescribed by the DSA/QOC approach.

Assessments of criteria against design options hold the greatest potential in terms of the explanatory content derivable from design rationale. Criteria represent the reasons why particular design options were adopted, discarded, or tabled for later consideration and so to the extent that captured criteria accurately reflect the reasoning of a design team, they provide a portal into the reasoning process. Identified Criteria were examined for their relation to elements of the *why?* explanation framework developed in previous chapters (see Chapter 5, Table 5) and then classified according to the *why?* explanation concept that appeared to best match the nature of the Criterion. The results of this classification are shown in the table below.

Table 8 - Why Explanations and QOC Criteria

Explanation Content Type	QOC Criteria
D-N Explanation	14/11% Criteria based on the constraints (laws) imposed by the underlying technical architecture of the system, e.g., the characteristics of the HTTP protocol, and the performance limitations of today's computers and software. Also criteria that arose from the need to conform to standards and legislative statutes, e.g., data privacy.
Functional Explanation	108/89% Criteria that related directly to the functional requirements of the system, e.g., use scenarios, desired outcomes from use, and system usability factors.
Pragmatic Explanation	-/- Pragmatic explanations can not be localised to particular Criterion without some knowledge of the context in which the explanation is being requested. In this sense pragmatic explanation is a meta-concept that describes the explanation content retrieval *process* in general and the explanation content product only when the explanation context and the explanation seeker's motivation are given.
Rational Choice Explanation	-/- Rational choice explanations are not localised in a single criterion, but are provided based on the set of Option-Criteria pairs that exist for a given design Question.

As seen in the above table, functional criteria were by far the most prevalent in the captured QOC, though some D-N laws, in the form of well-established universals and the constraints of the design materials (personal computers and software), did play a role in the design deliberations. Functional explanations based on design criteria may

seen as purpose-based, for example, a decision may be made to realise an identified system use scenario, or serve the goal of minimising development time. On closer examination however, many such criteria can be viewed as serving a particular purpose *within a set of constraints*, for example, minimise development time given that, as a rule, available time, money, and other resources are in most cases severely constrained. Similarly, a criterion such as "simplify the user interface" may be couched both in terms of the functional purpose served by the feature, in this case the user interface element, and in terms of a law-like statement such as a difficult to use interface will decrease the propensity to use, a statement that is supported by empirical research. The emergence of pervasive functional design criteria that act as design principles, possibly with the underlying status of laws, is discussed further in the section on meta and bridging criteria below.

Elements of a the pragmatic theory of explanation are impossible to identify without knowledge of the context in which the element is requested as part of an explanation. However, two of the central elements of the pragmatic theory are represented as meta-concepts in the QOC framework. *Contrast classes* are provided by showing the different Options identified in relation to a given design Question and the reasons (criteria) why one was selected over another. The second pragmatic concept, *relevance relations*, are also present in the QOC in that any factor (Question, Option, or Criteria) identified as relevant to the design or in some way affecting the form of the design, may be seen as 'naturally' relevant to the resulting artefact.

Rational choice explanation was also identified as supervening upon the entire QOC structure rather than isolated in the content of design criteria. By providing a view on the decision process followed by the design team, explanation requestors are potentially able to re-create the reasoning behind a particular design feature. This ability is only a potential however, in that as suggested by the results, some part of the design reasoning and design realisation process remains implicit even when attempts such as this are made to capture the entirety of the design process.

As highlighted by Shum (1991), criteria weightings can improve the expressiveness of the QOC notation by providing additional information about how criteria affected design decisions. Though functionality was added to the Drust application to capture criteria weightings, no explicit weighting were identified in the design deliberations.

Even in cases where explicit consideration was given to a pair of criteria related to a design option, for example, *"this option would make the system easier to navigate, but would severely complicate the programming"* the actual weighting of the criteria was always implicit in the deliberation. None of the assessments captured for criteria against design options included a criteria weighting that could be quantified in even simplistic terms.

Meta & Bridging Criteria

Meta-criteria, common criteria that transcend individual design questions and that underlay the bridging criteria applied to specific design questions, appear to play a key role in the final form of a design. The importance of these meta-criteria is shown in the VentureQuery project case, where 85 of 122 (70%) criteria were instances of a higher level meta-criteria. Meta-criteria have been described as the overarching principles or themes that guide design problems in a given context (Shum, et. al., 1993). The meta-criteria identified in the VentureQuery case appear in the table below along with the number of bridging criteria relating them to specific design questions.

Table 9 - Meta-criteria & Bridging Criteria Counts

Meta-criterion Name	Bridging Criteria
Maintain project scope	0
Use control data to minimise programming	0
Tailor questionnaire to individual users	0
Prevent user errors	0
Ease of system administration	0
Use familiar user interface components	0
Support future globalisation of the system	0
Prioritise generalisable requirements	0
Conform to the system metaphor	2
Make the system fun to use	2
Make the user interface aesthetically pleasing	2
Protect user privacy	3
Use scalable user interface components	3
Make the game interesting	3
Maximise runtime performance	6
Maximise educational value for users	7
Easy to build questionnaire	11
Flexibility of use	14

Meta-criterion Name	Bridging Criteria
Easy to use questionnaire (respondent)	16
Simplicity of construction	20

Meta-criteria were captured both as part of specific design-related questions and from discussions of the higher-level principles guiding the design and development effort. Meta-criteria appear to play a key role in the evaluation of design options and help to highlight the central role of a relatively small number of concepts and their trade-offs, e.g., system flexibility versus simplicity construction, in the final form of the system design. Simplicity of construction and the minimisation of resource utilisation afforded by this simplicity was the most prevalent meta-criterion for the project. In many cases design deliberations centred around the basic tension between a more flexible and functional design that is easy to use, and the relatively high costs of implementing features to support these goals.

Assessments

Assessments describe the content of the relation between an Option and a Criterion in a QOC structure. Both the term, i.e., whether the Criterion is considered favourably or unfavourably (+ or -) in relation to the Option, and the underlying arguments that give rise to the term comprise an Assessment. From a theoretical perspective, Assessments hold the potential to provide a wealth of explanatory content regarding the form of a design and any resulting system. The results of the VentureQuery case show a paucity of explicit assessments emerging from design deliberations, only 114 of 436 possible assessments (Option-Criterion pairs) were identifiable in the design deliberations.

6.3.2 Explanation Delivery with QOC and *Drust*

As described in section 6.2.4, the Drust servlet extends the JavaHelp architecture to provide HTML, hypertext-based explanatory content, using the QOC design rationale as a knowledge base for this content. As discussed in Chapter 2, much of the prior research into integrated explanation facilities has focused on the construction and delivery of 'natural' explanatory dialogues that attempt to mimic the explanations provided by experts to those less adept in a given domain. Drust uses a fundamentally different explanation mechanism, relying instead on the user to construct the explanation they deem most appropriate for the current scenario given a set of

explanation content components hypothesised as sufficiently comprehensive. In the Drust model, these components are delivered as hypertext, an information design paradigm now familiar to many, if not most, users of computers and information systems given the rise and current prevalence of the Internet and World Wide Web.

The idea of using hypertext as a delivery vehicle for design rationale retrieval is central to much of the prior research into design rationale use and usability (Shum, 1991; Fischer, et. al., 1996). The hypertext model maps particularly well to the structure of a QOC-based design rationale in that QOC is not circular or recursive and it is modelled as a relatively shallow hierarchy or graph (four levels as question-option-criteria-meta-criteria or as question-option-criteria-assessment, five levels with the Drust-specific QOC outline added). The shallow, directed nature of a QOC structure and the hypertext that maps to it makes it relatively easy to comprehend and to navigate. Patrick and McGurgan (1993) argue that if the information provided by an online help system exceeds four levels deep, designers should consider reorganising their information to meet this limit. As shown in the figure below, a fully-expanded QOC structure in Drust can be displayed in a single page window.

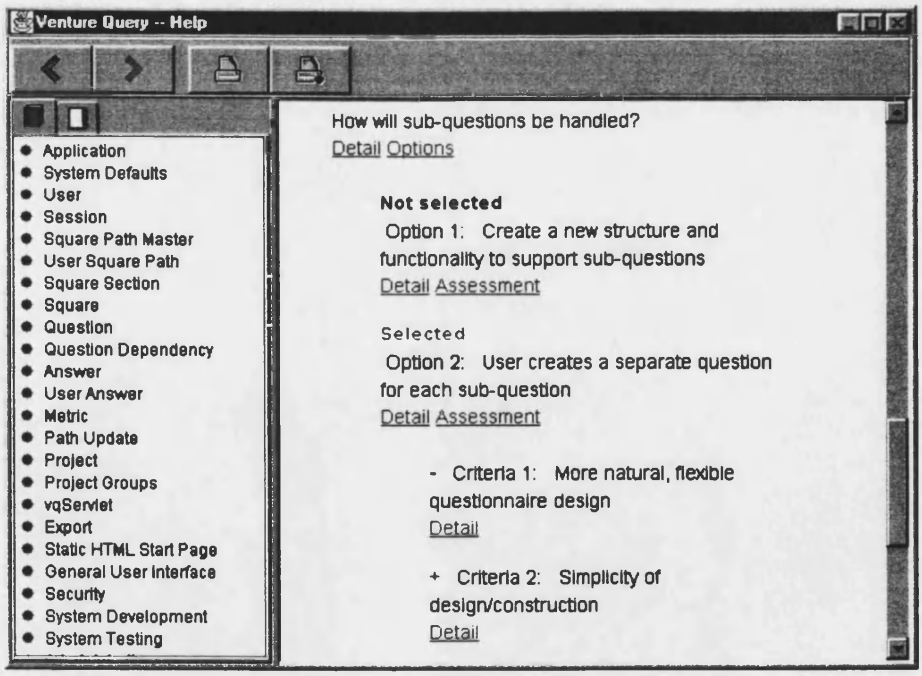


Figure 30 - Fully Expanded Drust Explanation

Two methods of providing help to end users are application-directed and user-directed (Roesler & McLellan, 1995). Application-directed mechanisms attempt to determine when the user needs help and then actively provides it, user-directed is a passive

model where users decide when they need help and then actively seek it out. The latter approach, adopted in Drust, also conforms to the minimalist model of technical communication, discussed in Chapter 2, which suggests that users should be provided with just enough information to let them explore and discover on their own (Manning, 1999). However, a key problem with this approach is deciding which information to present to the user first (Draper, 1998). The approach taken in the VentureQuery case study and embodied in the Drust information design was to map these entry points to the major concepts in the system domain, in the form of QOC Outline components. In an object-oriented software development project, the domain concepts generally map to the classes designed and programmed in the system. This mapping between domain concepts, the components of the software, and the design rationale content helps to provide structure to the points of entry into the design rationale explanation base. In this way the Drust model potentially overcomes some of the problems identified with explanations that rely on 'canned' text, where the granularity of this text relative to the modularity of the software components it explains can be difficult to maintain (Swartout, 1983). Explanatory text fragment granularity must correspond to the granularity of the underlying software so that recombination of software modules does not result in the loss of relevant explanatory text or, conversely, in providing irrelevant text.

An important potential advantage of the Drust approach is that although the QOC-based help available at a specific point in time is in a sense 'canned', both the standard help content and the explanatory content available to a target application can be updated at any time through the addition of new information to the knowledge base underlying the application. The information in such updates is immediately available to the target application, which uses a central, HTTP accessible DBMS as its information source (note however that an Internet connection is required to retrieve this information from the DBMS). Information systems have been characterised as 'living' artefacts (Paul, 1994), never frozen but constantly evolving to meet new use scenarios and the local requirements of where they are implemented (Bowker and Leigh Star, 1994).

There exists a degree of polarisation over the best direction for innovation and progress in HCI research. Some researchers argue for more investigation into dynamic, adaptive interfaces; others argue for a focus on the industrial design of

software, for example, through improved use of metaphor, as the way forward; a third group of researchers has emerged who argue the veracity of a mixed mode of HCI research (Horvitz, 1999). These mixed-mode initiatives attempt to identify the best combination of direct manipulation on the part of the user and automated prompts on the part of the system. One product of mixed initiative HCI research is the feature of Microsoft Outlook where incoming electronic mails are scanned and if they include content that appears to be, for example, a meeting request, the system prompts the user with a completed appointment record based on the people, dates, and times mentioned in the message. The user may then choose to discard the appointment if the system's 'guess' turns out to be wrong.

Wenger (1987) argues for the need to be open-minded when considering new opportunities for person-machine knowledge communication, not limited by the paradigms that exist for human-to-human communication. While humans appear to excel at the use of perception, adaptability, and creativity in communication; computers seem to have the edge in completeness and consistency. Chapter 2 discussed how attempts to manage explanatory dialogue with users of an expert system are necessarily fragile since they rely to such a large degree on the user to follow a dialogue plan that has been conceived in advance. An argument from the expert systems literature claims that procedural explanations are deficient because they rely on determining in advance how a given piece of information will be used (Lameberti & Wallace, 1990). The minimalist approach to information design, as well as research into the nature of learning in a community-of-practice suggest a foil to explanation dialogue management, learners do not want to be led through a dialogue plan, they want access to quality content that they can explore and understand on their own and within their peer group, in their own context of use.

6.3.3 Costs and Potential Benefits of QOC Capture

Audiotape capture and transcription of design meetings is a relatively inexpensive overhead for a software development project. High-quality recorders and the tapes they use cost about £100 and £2 respectively at the time of writing. Professional audiotape transcription services range from £8 to £15 per hour with transcription times averaging three hours for a 90 minute cassette. However, subsequent creation of well-formed and useful QOC structures represents a significant additional cost.

Creation of finished QOC structures from transcripts represented another 3 to 5 hours per meeting transcript, depending on the density of the meeting's design content, and this task almost certainly must be performed by a development team member with intimate knowledge of both system design concepts in general, and the particulars of the individual project. Such experts are expensive and their time is in great demand from all of the stakeholders in a large IS development project (Curtis, et. al, 1988).

The Drust system itself represents a relatively low-cost, low-complexity solution to the problem of delivering design rationale-based explanations to almost any software application. The software was itself developed using primarily free, 'open' software tools including Java, the MySQL DBMS, and the JRun software bridge that allows Java servlets to be run as extensions to a web server such as Microsoft's IIS. A Java IDE, JBuilder3 Professional from Borland, was however used as the primary development environment, but is not required for runnable instances of the application.

A potential by-product and benefit of the approach adopted for the VentureQuery project is that good content for both print documentation and standard online help systems is obtained during the analysis and transformation of the design transcripts into QOC. Especially rich in this respect are cases where someone from outside the design team, a reviewer or prospective user for example, sits in on a design session and continually requests clarification of the various features of the system. These clarifications were in many cases able to be used almost verbatim and transferred directly to the standard help database.

The primary potential benefit of the Drust approach is as a tool to facilitate enhanced understanding by both system developers, other project stakeholders, and, most importantly, the users of applications that implement the system. Some theoretical research suggests that users of IS will not use hypertext–based explanations because the cognitive effort involved in actively retrieving them (Gregor & Benbasat, 1999), but others argue that hypertext-based learning systems hold great promise, though little research has been done on specific mechanism that may be employed in this context (McKerlie, et al., 1993). The Drust architecture holds the potential to derive the benefits identified in this latter research corpus, which includes work in the minimalist, constructivist learning tradition, but more ecologically realistic,

longitudinal research is required to determine whether these benefits are actually obtained using the approach described here. This and other future research requirements are discussed in more detail in the final chapter that follows.

6.4 Discussion

This section discusses some of the most salient findings from the study and relates these to theoretical concepts reviewed in previous chapters. The next and final chapter, Chapter 7, integrates the findings from this study with the findings from Study 1, discusses the implications of these findings, and identifies areas for further research.

Capturing Complete Explanations

The most obvious and important result of the VentureQuery case study was that much of the design deliberation, including the crucial assessment of criteria against design options as well as the actual process of deciding on elements of a final design, was not made explicit in the putatively collaborative process of design. This finding lends credence to ideas espoused by design theorists since at least Schön (1983) that deliberate techniques must be applied in technological design in order to promote explicit consideration and reflection of the design problem solving process. One way in which this ideal may have been achieved in this case was through the use of Design Space Analysis (DSA), the process that QOC's developers conceived to direct use of the notation in design practice.

Constructing a retrospective, QOC-based design rationale from a project where team members did not follow the DSA approach contradicts many of the guidelines and heuristics that have been derived from prior DR research using QOC (e.g., Belloti, et. al, 1991; Shum, 1991; MacLean, et. al., 1996). Though a retrospective approach to DR capture does help to work around some of the design process disruption associated with integrating DSA/QOC into a project 'ecology' (Buckingham Shum, et. al., 1997), in the context of explanation content capture the costs of not following the approach appear to be too great. Though retrospective DR has been used effectively to *evaluate* designs of different systems (McKerlie, et al., 1993; Shum, et. al., 1993), in the VentureQuery case the lack of a DSA-derived process means that deliberations on a particular design issue did not always result in a 'proper' QOC

structure, with, for example, multiple Options generated for each Question, and each Criterion explicitly applied to the evaluation of each Option.

Many of these completeness issues that arose in the use of QOC may have been avoided if the DSA approach had been applied more rigorously, if all project team members had been indoctrinated in its use, and if the QOC representations had been used to set the agenda from one design meeting to the next. As noted earlier, the DSA approach was not employed because of the anticipated negative effect of prescribing a particular process template on the design process. However, it is likely that the DSA approach would have forced subtle and sometimes unstructured deliberations into focus and significantly eased the identification and construction of QOC elements in the Drust knowledge base. Presumably, this would also enhance the explanatory power of the approach by providing more complete deliberations related to a given design concept.

One way that the VentureQuery design rationale could have been made more complete is if the QOC creator and/or the application programmer implementing the design (as reported here, the researcher in both cases) had filled in gaps in the captured design rationale as they were identified and resolved. Little research has addressed the issue of how a solo programmer might use DR in the course of working through a development problem (Conklin & Begeman, 1988). However, in the interest of the research goals guiding this study, in particular the question of how much explanatory content can be gleaned from a relatively unprescribed, naturally occurring design process, this approach was not adopted.

There seemed to be clear value in performing the QOC transformation after each design session rather than fully retrospectively as was done here. This would have supported a much more iterative and comprehensive design process. Also, capturing rough QOC as soon as possible after each design meeting may have facilitated the QOC 'scribe' filling in some of the gaps in the discussion and would resulted in a far less overwhelming analysis task than a full retrospective DR being constructed at the end of the project. However, even in the 'impoverished' way that QOC was used in the case, it still provided a beneficial framework in which to document design issues. Even when only one Option was available, or one Criterion, the formalism still

expresses what the designers were thinking, even if their thinking was incomplete and very limited in its scope.

Though it has been claimed that the most significant issues in any software project are discussed in design meetings, rather than informal discussions or not at all (Kuwana & Herbsleb, 1993), it is also possible that certain implementation decisions are made in isolation by individual members of the project team, and therefore never deliberated and never recorded as part of the rationale (Fischer, et al., 1996). Such 'rationale ambiguity' may be an unavoidable, even essential characteristic of technological design and construction (Bowker and Leigh Star, 1994). If large, complex design and development projects are to be completed within their inherent resource constraints, not every decision and relevant factor can be deliberated and the challenge becomes one of defining an acceptable level of ambiguity rather than eliminating it altogether. It is important however that an explanation facility that relies on potentially ambiguous knowledge make clear to its users where its explanations are possibly incomplete or imprecise. The Drust model helps to expose such ambiguity through the use of a simple model for explanatory content, QOC, where missing nodes and links in the structure are relatively easy to comprehend.

Relating the structure and function of a tool to the domain of intended use is a necessary component of documentation and other explanatory materials designed to increase device understanding. Some obvious questions are however, how much is enough and when do we stop (Draper, 1998). It has been argued that DR techniques need not be applied to every design problem or issue, only those that require more careful and deliberate analysis (MacLean, et al., 1996; MacLean & McKerlie, 1995). To support complete and comprehensive explanations, DR information would need to exist for every feature in the system. Though Rettig (1991) claims with reference to the minimalist approach to documentation (see Chapter 2), that a central goal of software documentation should be "...to give the learner more to think about, but less to overcome", Fischer (1999) argues that IS information needs to be available to support "learning-on-demand" scenarios where deep information is not forced upon, but made available to project stakeholders.

Another issue that may have detracted from the completeness and quality of the final DR knowledge base is that a large proportion of the text captured from the design

meetings related to the domain, specifically to the domain of venture capital seeking and the kind of questions and answers that might be helpful to a novice entrepreneur. This is a complex domain and the project team included no one with expert knowledge of the subject area. This meant that a great deal of time was spent researching and then discussing the domain and the potential role of the application within it. As the project domain evolved from a domain-specific to a more general electronic questionnaire building and publishing tool, the content of these discussions become increasingly less relevant to the form of the final system design. Domain-specific discussions were not captured as part of the design rationale unless they were part of a scenario that was being used to expand or evaluate a design object.

Explaining with QOC

Strictly speaking, the Drust model provides system users with explanatory content, not the explanation itself. Following the minimalist, constructivist approach to learning, users are expected to explore the explanatory content presented to them as hypertext, and form their own explanations relative to their scenario of use and their specific use context. Jacobsen (1992) has argued for the application of the architectural discipline known as "wayfinding" to the field of human-interface design. Wayfinding is concerned with providing a context within which the structure of a building can be more easily interpreted and therefore, understood. Architects use wayfinding techniques to design spaces that people can intuitively navigate (Passini, 1984). The aim is to minimise intrusive directives and to promote the use of subtle cues to help make the 'user' of a designed space feel more comfortable and at ease. The increased use of metaphoric icons in modern user interfaces, such as the desktop and toolbar palettes, is an application of these techniques. The node-link structure of hypertext is another, albeit more abstract, example of how this approach to wayfinding has been implemented in 'information space'.

Vannevar Bush, Director of the U.S. Office of Scientific Research and Development during the Second World War, first described the need for a mechanism to chain together information into a coherent and easily accessible body of knowledge (Bush, 1945). His article was both a call for a pause to reflect on the successes of the scientific establishment during the war, and an insightful analysis of the need for better mechanisms to manage the enormous body of knowledge being generated in

fields of science and technology. Bush argued that despite the volume of knowledge being generated and advances in communications technologies that allowed it to be rapidly disseminated, little progress had been made on technologies to improve the use of this knowledge. He puts it best:

> *"The summation of human experience is being expanded at a prodigious rate, and the means we use for threading through the consequent maze to the momentarily important item is the same as was used in the days of square-rigged ships."*

To address this problem, Bush proposed (foretold?) a device, the "memex", a personalized information management system designed to include all of the books, records, communications, etc. of interest to the user. Central to the design was a coding system used to link related pieces of information from the main store into an associative index. The associative index on a given topic represented the user's "trail" of interest through the subject matter. These trails were stored and any time that the user accessed an item integrated into the trail, the entire trail was made instantly available for review.

The associative indexing technique that Bush described is now one of the most pervasive design models of the information age, hypertext. Though clearly progress has been made in the areas of computer performance and storage, database management systems, and the Internet, among many others, the recent explosion of interest in the field of knowledge management demonstrates that the problems involved in a managing the quantity of information generated by modern society and its technologies has yet to be resolved, including how to best capture, organise, and retrieve the mass information and knowledge that leads to the development of a complex information system.

As Carroll (1997) points out, despite the continued role of the 'waterfall' model of software design and development, in reality the software development process is characterised by a much less structured process of problem and sub-problem identification, explication, and specification. Surprisingly one problem that emerged was one of explanatory completeness at the level of *what* questions. Some design options appeared to emerge almost magically from the design discussions in the sense that it was often difficult to see the chain of reasoning that led to a particular design question being deliberated or design option being proposed. This problem was

exacerbated when a design option took the form of a high level design object, for example a class, and then candidate object components were enumerated in rapid succession. Even more problematic were cases where the design team focused on the relationships and dependencies between system objects, and by reference their corresponding QOC structures. At first glance, such cases would appear to have generated a rich set of rationales but upon closer analysis the conversation moved so quickly between foci that much of what is required to populate the QOC was found to be missing. This highlights the potential role of the assumptions made between team members about their grasp of a design issue, and once again highlights the potential benefit of a reflective design process, such as DSA, in helping to make these assumptions explicit.

There was an apparent asymmetry with respect to the amount of discussion allocated to certain features over others. This asymmetry was especially acute with respect to *what* questions versus *how* questions. Relatively little discussion was provoked by the identification of a new candidate *entity* for the system, while discussions of new *processes* more often evoked long discussions. This seemed to lead in many cases to the inclusion in the design of system entities that were poorly defined and poorly understood outside of the context of the processes in which they played a role. This seems problematic in the context of object-oriented design, where the generation of a complete justification and description for a given entity can assist with the creation of more modular system objects, with more well-defined semantics and behaviours. Certain more complex problems, such as the process of managing the path through the questionnaire, elicited a significant amount of deliberation spanning several design meetings, much of which appeared redundant when analysed in the meeting transcripts. However, much of this redundant deliberation may serve the purpose of increasing team understanding of more difficult concepts, even if no new information (including Questions, Options, or Criteria) appears to be generated.

Scenarios

The use of scenarios to both envision design features and to evaluate new features was pervasive in the design transcripts. The crucial role of scenarios in IS design has been highlighted in many works (see Carroll, 1995 for a collection) and use scenarios have been identified as a key source used to generate Criteria for an evolving design

rationale (MacLean & McKerlie, 1995). The latter work describes how Criteria in the DSA/QOC framework can be used to relate tasks or use scenarios to the DR, a Criteria represents the task, and an Option represents the specific system design feature which supports the task. Because in the VentureQuery case domain analysis proceeded in parallel with design, design sessions often involved using some newly acquired domain requirements - in the form of the questions, possible answers, and metrics for evaluating those answers that might be employed by venture capitalist - as the basis for a logical test of, for example, the attributes and behaviours of a class or interrelated classes. The importance of scenarios in the requirements identification and management process was highlighted by Curtis, et al. (1988), who found that when the scenarios that drove requirements definition were abstracted out of the documentation, project team members later found it difficult to understand the meaning and context of requirements.

It appears that a shared understanding of the evolving design emerges from an iterative (and redundant) process of discussion involving what MacLean and McKerlie (1995) identify as *evaluator* and *envisioner* scenarios (see Chapter 4). Certain complex problems resulted in the generation of many such scenarios and appear to play a role in providing the development team with their rich, shared conception of the problem space and the form of the artefact relative to the constraints presented by this space. However, capturing this depth is time consuming and tedious, and it is sometimes unclear whether a given deliberation is truly redundant or is the process of forming a new conception of the problem space. Transforming scenarios into structured QOC was often awkward given that they frequently involve consideration of the interrelationships of many design objects. For example, in the case of VentureQuery the basic process model for the system emerged from the abstract scenario: "Ask a Question, get an Answer, apply a Metric, give a Score." This abstract scenario was then deliberated relative to the evolving system model by applying both envisioner and evaluator scenario snippets and considering whether and how the attributes and behaviours of the system objects would realise this scenario.

Mapping System Objects to Design Rationale

A significant issue was the need to capture the relationship between a particular QOC structure and the design object to which it relates. This was required to facilitate

retrieval of the QOC-based explanation based on a request related to a particular feature of the system. The challenge of relating design rationale to artefacts under construction has been previously identified and discussed in prior research that has attempted to embed DR approaches in the software engineering and device engineering contexts (Fischer, et. al., 1996; Gruber & Russell, 1996). One of the critical success factors to an IEF such as the one described here is that it manage to relate different pieces of information from the knowledge domain and the artefact design (Patrick & McGurgan, 1993). In the domain of online helps systems, the advantages of help content that is mapped to system objects, as opposed to monolithic help sets for large systems, includes both more relevant content and more portable software components (Chamberland, 1999). However, the problems that arise in attempting to create help content schemata to organise componentised help are significant, especially with regard to the interdependencies of most software system components, and worsened by the addition of design rationale information to the help data set.

Even more problematic was the need to track all of the factors that influence the form of a design object as it evolves through the system development lifecycle. Knowledge-base maintenance such as this has been identified as one of top cost components in expert system projects (Hayes-Roth & Jacobstein, 1994). As system components evolve, they become more abstract to handle an increasingly broad range of use scenarios. The more generic the abstraction, the fewer the system components and therefore the simpler the design. As the components become more abstract however, they become less domain specific, less well related to actual entities in 'the world', making them increasingly more difficult to understand for those without knowledge of how this process of abstraction facilitates the software development process.

A related finding was the importance of *control data* to the explanation of IS functionality. In order to make them both more abstract and more user-configurable, modern software systems rely heavily on system parameters, or switches, to support the creation of customised instances of a given system for a particular use context. Clancey (1986) found that strategic information crucial to effective explanation is often embedded in the implicit relations between such control data parameters and their values. For example, in VentureQuery the behaviour of a particular online

questionnaire system is substantially determined by the control data entered by the user of the questionnaire builder sub-system. The more flexible the underlying system that uses these parameters is, the more use scenarios are supported and the more difficult it is for the design team to capture the information needed to explain the operation of the system *in the particular use context.*

QOC seemed at first ill-suited to capturing the architectural elements of the system. Especially problematic were those that were initially generated as lists or enumerations of possible features (system classes) in response to a particular design question, for example "what are the components of a single questionnaire page" that appear early in the design process. Such deliberations were captured by creating lists of Options under the design Question during the rough QOC capture stage. However, it appeared clear that deliberations such as these should be mapped to the system object under discussion as soon as this relation was understood to exist. The approach taken to address the problem of relating design rationale to related system components involved the addition of the QOC Outline to the Drust DR capture model, as described earlier in this chapter. The QOC Outline represents a kind of ontology for the system architecture in that it defines the classes that make up the architecture in terms of the design rationale that justify their existence. Identification of new Questions and Options generally initiated the creation of a new candidate class, if an appropriate one did not already exist, and were immediately categorised by reference to a system component. This approach was effective in helping to relate system objects to the requirements that caused their creation and then to the design process that led to their ultimate form (or rejection as part of the system). It also provided a relatively simple (possibly simplistic) categorisation mechanism to facilitate retrieval of the QOC during system use.

7 Discussion and Conclusions

This final chapter first integrates the results of the studies described in the previous two chapters, 5 and 6, and explores their relation to the theoretical framework developed in Chapters 2 through 4. The second section discusses the limitations of the study and the third identifies areas for further research that emerged from the work. The fourth and final section concludes the dissertation.

Chapter 2 reviewed prior work in the broadly-conceived domain of IS explanation including both the content and delivery perspectives in system user manuals and help systems, intelligent tutoring systems, and expert system explanation. Of particular interest is Clancey's (1983) structure-strategy-support model of expert system explanation and two important insights derived from this work. The first of these is that the process of translating a domain strategy into a system structure represents a significant component of the explanation content needed to explain the resulting system's operation in the domain. The second insight is that support knowledge may help to provide this information by justifying the existence of a system process or feature relative to domain facts or rules. The essence of this thesis is the development of an ontology for this support knowledge and an investigation into how this knowledge can be captured and delivered in an IS integrated explanation facility.

Chapter 3 describes the construction of an initial conceptual framework, or ontology, for IS explanation content based on a review of the philosophical literature on theories of explanation. This review was motivated by the fact that notions of explanation content and structure in the domain of IS generally lack theoretical grounding, typically dictionary definitions of what constitutes an explanation act as the starting point for this work. Because philosophical investigations into the nature of explanation generally focus on idealised, necessary and sufficient constructions, and because the theories reviewed represent enduring ideas in the field, it was hoped that they could help to provide the foundation for a more conceptually rich content framework. An ecumenical approach was applied to the analysis of these theories and a key criterion for the adoption of a concept from any given theory was the potential to provide explanatory power in the domain of human-engineered information systems that emerge from complex organisational and social contexts.

Chapter 4 reviewed concepts and techniques from the field of design rationale and proposed design rationale as an approach to capturing the support knowledge, the *why*, needed to provide explanations of an IS form relative to its domain of use. Design rationale is a meta-theory of the design process as well as a collection of notations and tools for capturing, structuring, and providing access to the components of an artefact's "design space" (MacLean, et. al., 1996). The essence of design rationale is that of design-as-argument; and of these arguments as a knowledge base to be used in support of the design process. The work reported here is an attempt to extend design rationale's utility to the domain of IS use and users, a potential which has been discussed but not investigated in prior research.

The product of the review and analysis described in Chapters 2 through 4 is *DREX* (Design Rationale Explanation), a conceptual framework of IS explanation content and a proposed approach to capturing this content as depicted in the figure below.

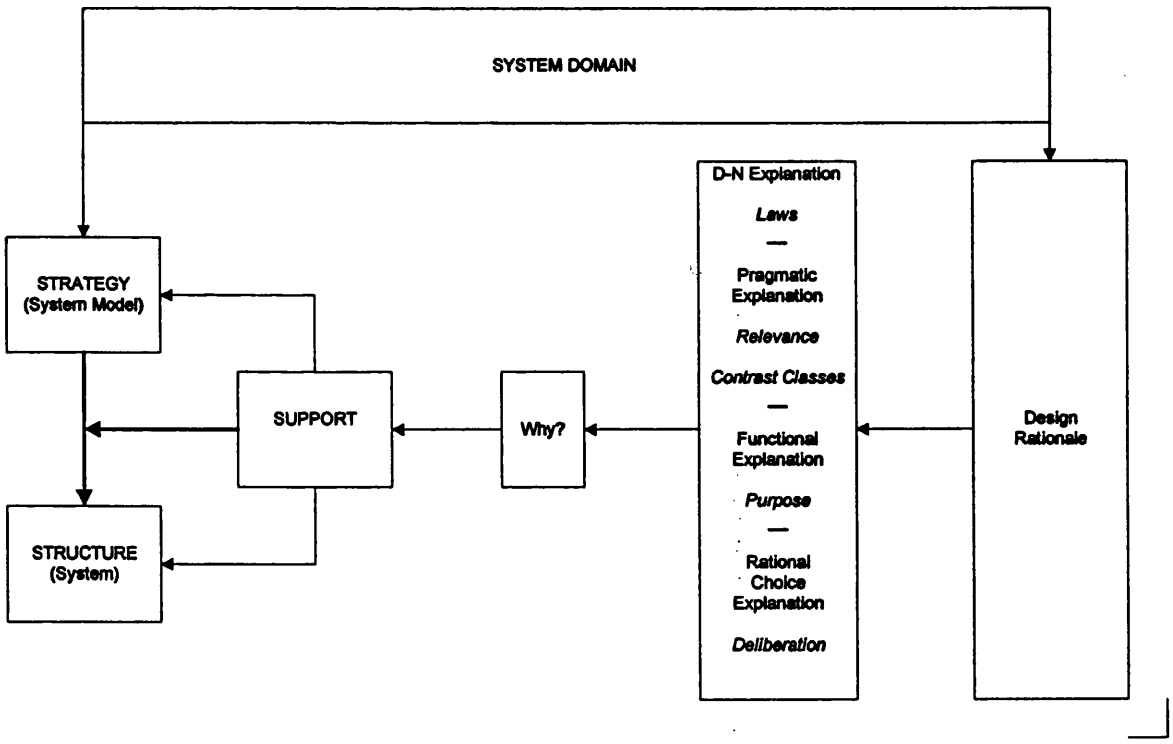


Figure 31 - A Framework for IS Explanations based on Design Rationale (from Chapter 4)

Chapters 5 and 6 describe the empirical components of the thesis, the methodologies and procedures assembled to investigate the DREX framework and the results that were obtained from these investigations. The study described in Chapter 5 involved a series of semi-structured interviews with IS development professionals. The goal of

these interviews and the subsequent analysis of their results was to examine the viability of DREX relative to the perceptions and experiences of IS development professionals in the field. The second study, described in Chapter 6, was an IS design and development case study. The goal of this study was to employ the DREX framework in an IS development scenario by capturing design deliberations, constructing a retrospective design rationale from these deliberations, and providing access to this design rationale from within the system that was constructed from the design. Two proof-of-concept software systems were developed as part of this case and the issues that arose in their construction form a component of the reported results. These two studies are reported in detail in Chapters 5 and 6, and their results are integrated and interpreted further in the sections that follow.

7.1 Integrating the Findings

Information systems emerge, evolve, and are used within a complex network of factors. These factors may be technical, psychological, or social in nature, and may be causal or intervening in their effect. System projects are affected by corporate or institutional politics, procedures, and culture (Curtis, et. al., 1988). Prior research suggests that answers to *why* questions that draw on this knowledge may assist in reconstructing the context for those potentially unaware of the range of factors that describe the environment in which the original developers worked (Kuwana & Herbsleb, 1993). Explanation of an information system and its features and processes should include access to the range of factors that affect its evolving form during system analysis, design, construction, testing, and use. These factors and their effects constitute a system's design rationale, which can potentially contribute to IS explanations that are deeper and possibly more useful than those commonly provided in user manuals and help systems.

The idea that system usability and usefulness may be enhanced by narrowing and making more transparent the gap between the system development and use contexts is not new (e.g., Norman, 1986; Carroll, et. al., 1994; Redmiles, 1993), and the research reported in this dissertation is meant to contribute to this body of work by exploring the dimensions of explanation content and how this content might be sourced in the analysis and design process.

The theoretical and empirical work reported here makes the following novel contributions to knowledge in the fields of IS explanation and design rationale use:

1. A theoretically grounded framework of IS explanation that draws on the extensive philosophical literature rather than simplistic, dictionary definitions.

2. A set of empirically derived weightings that suggest the relative importance of different elements of the explanation framework from 1.

3. A set of empirically derived critical success factors for the process of design rationale capture, representation, and use.

4. Capture and analysis of the design rationale for a complete, realistically complex IS development project.

5. An analysis of the design rationale relative to the explanation framework derived in 1.

6. It describes two proof-of-concept software tools for capturing and delivering IS explanation content based on design rationale.

The three sections that follow integrate these contributions and analyse them further in terms of their relation to IS development and use theory, to prior research in these areas and to state-of-the-art practice.

7.1.1 An Ontology of IS Explanation

Explanation of a complex technological artefact is a multi-faceted information construct, and development of a precise framework for a necessary and sufficient explanation in the domain of software information systems may be impossible without prior knowledge of all scenarios likely to motivate an explanation seeker. Nonetheless, development of a framework that attempts to describe what is necessary, and supports the individual user in determining what is sufficient for a given scenario may still provide real benefits in the contexts of IS development and use. Parnas and Clements (1986, see Chapter 4) argue that striving for an ideal, rational framework in the domain of software development and use makes sense even when one seems unobtainable in practice because of the guidance such frameworks provide in steering

development efforts towards improvement upon *ad hoc* approaches. The same philosophy applies to the work reported here, even if production of ideal, user-tailored explanations is beyond the capabilities of today's software and computing machines, providing IS users with more complete, rationalised explanatory content along with an access mechanism that facilitates use of this content may help resolve differences in the user's understanding of a system, and the system's understanding of a user.

Most research into system explanation is process oriented in that it focuses on human-to-human explanation and attempts to replicate this process in a software mechanism. By contrast, this thesis has focused on the nature of explanatory content, on the structure of and relations between different types of content, and on a proposed process (design rationale) for sourcing this content. Very little is known about the kinds of knowledge people use to perform actions or how that knowledge is employed (Draper, 1998) lending weight to the suggestion that explanation research should shift in focus from the explanation process to the content provided in system explanations (Clancey, 1993). The work reported here progresses this shift by grounding ideas of IS explanation in the philosophy of explanation and by examining these ideas empirically using input from experts in the practice of IS development and use.

The initial framework of IS explanation content derived from the philosophy of explanation literature was constructed through an ecumenical approach to these theories. Each theory was considered in light of the special nature of IS as a science, at a basic level (i.e., hardware) the function of an IS is constrained by the same laws of nature that bound other physical devices, but other factors play an important role in the final form of the device. In particular, the form of an IS is highly determined by the function it is meant to perform in its domain of use, but the relative lack of material constraints in IS design along with the somewhat arbitrary and serendipitous deliberative process that seems to characterise IS development means that the form that emerges from a particular set of functional requirements has many potential realisations. The research reported here considers the nature and importance of these different factors given their potential role in helping to explain these realisations.

Study 1 reported in Chapter 5 began with the initial theoretical framework derived from the philosophy of explanation literature and examined this framework relative to the perceptions of IS development professionals. Though this study examines only

one of at least two important dimensions of the explanation equation, that of the explainer, it makes a contribution to the study of IS explanation by providing a framework grounded in **a)** perceptions of what explanation content is relevant given in-depth knowledge of a system's functionality **b)** perceptions of the content that appears most useful to users of these systems based on prior experience with fielding these queries **c)** perceptions of what is actually possible to capture and provide given constraints on the IS development process.

The DREX framework that emerged from the study consists of a two-level perspective on IS explanation. At the first level, the front line of explanation content consists of *operational* explanations that provide system users with identification of system features (*What does it do?)*, basic information about how to use these features to perform a task (*How do I use it?)*, and content describing some of the underlying structure and relationships that help users to understand how individual system components are assembled to support sometimes complex, multi-part processes (*How does it work?).*

The information provided by these operational explanations bears little relation to philosophical theories of explanation, which are primarily concerned with answering *why?* questions, but reflect the roots of at least two of the most prominent schools of instructional design: the *systems* model and the *task* model of IS instructional content. Chapter 2 described how the systems model is concerned with providing a hierarchical view of instructional content that maps to the structure of the system itself as a series of, for example: menus and menu items, windows, fields, and buttons. The task model takes a more holistic approach by documenting how one or a series of system features are used to perform some identified task. Each of these models has strengths and weaknesses (see Chapter 2) and both may be contrasted against a third model of instructional design, the *minimalist* model, which argues for a sparse set of instructional content that has been user-tested for its ability to support active, exploratory learning.

The more in-depth content characteristic of answers to *why?* questions represents a second level of explanatory content which can be viewed as providing backing or support for operational explanations. This explanatory content relates closely to the content identified in the philosophical theories. Deductive-nomological explanation

views laws, both the 'hard' laws of nature from the physical sciences and the 'softer' "well-established universals" that emerge from study in the social sciences, as central to why explanations in that they help to unify disparate facts under an umbrella of generality. At the other extreme of explanatory content structure, the pragmatic theory takes an 'anything goes' approach, in other words, that which is explanatory is that which is sufficient to impart understanding to any particular individual in the unique context that defines their explanation request.

Both of these extremes were represented by the data collected in Study 1, with the pragmatic approach emerging with approximately twice the conceptual density of the D-N model, and with the highest density overall. Concepts from functional and rational choice explanation emerged with approximately the same density as those of the D-N theory. Recall that the functional model relates theoretically most closely to explanation of designed artefacts, where it is supposed that a design and its elements take the form that makes them most suitable to the purpose that gives rise to the design effort. The rational choice theory explains by exposing how a design team employs some kind of utility function to each design deliberation, carefully weighing the pluses and minuses of each design question relative to both the purpose underlying the design and the myriad of constraints that bound its final form.

These results suggest that efforts to tailor explanatory content to individual users and use scenarios are well justified. The success of an integrated explanation facility may be predicated upon its ability to establish *relevance relations* (van Fraassen, 1988; 1991) between the context of the explanation request and the information available to respond to that request. However, computer-based systems are famously unaware of the full richness of their use context (Dennett, 1990). As discussed further below, the solution to this problem may lie in explicitly recognising the role of the human user in the making these relevance determinations, and in capturing and designing information to support this process.

In addition to providing empirical evidence regarding the relative importance of different explanatory concepts, Study 1 also examined the different factors that affect both the utility of these concepts and the viability of capturing associated content from the IS analysis, design, and development milieu. The results of this examination were inconclusive in terms of the general viability of the DREX approach to

providing explanations, both significant benefits and significant impediments were identified, but benefits and impediment were themselves an important outcome of the study in that they identify focal issues for further research.

Among the more well-represented impediments, participants often expressed the opinion that end users are simply not interested in explanatory content derived from the IS analysis and design process. An example in support of this line of reasoning is the World Wide Web, a complex system that has succeeded in the marketplace without the benefit of supplemental instructional materials (Draper, 1998). This view also lends support to the idea of a "production paradox", which claims that most users of complex IS are simply too busy trying to manage their workload to spend time learning more about the tools they use (Carroll & Rosson, 1987). Though expert users in complex domains already know how to accomplish tasks in their domain, the systems they use as aids may introduce new or different ways of accomplishing these tasks (Hackos, 1998). However, a deeper understanding of the form of a system relative to the domain may be required to realise this potential. Draper (1998) for example, argues that if people understood the concepts underlying the systems they use, many more would use this technology to help them in their work and that this barrier to knowledge results in an inability to relate a system concept to the problem domain, one of the keys to technology and device understanding.

Other negative factors that were identified include a lack of time to spend reviewing the design rationale content that might provide a richer a understanding of the IS. This factor is obviously closely related to lack of interest and the production paradox. Some study participants also felt that managing the competing interests of IS development project stakeholders was already too great a challenge without exposing, and therefore opening up for time-consuming debate, the intricacies of the design decision making process. The converse of this issue are the potential benefits gained through a more open, participatory design, discussed further below.

Another challenge identified was the potential difficulty of transforming analysis and design information so that end users, most of whom are presumably unfamiliar with the 'language' of system design, would find the information comprehensible. Finally, the idea of making an IS design public through design rationale-based explanations

was seen as a potential threat to the intellectual property represented by a complex design if made available to competitors.

In contrast to the factors that appear to diminish the viability of the DREX approach, study participants also highlighted a number of positive outcomes that could potentially result from its implementation. The most conceptually dense positive outcomes were enhanced participation by project stakeholders, better communication of the progress and form of an IS design project, a concrete organisational memory as a by-product of the design rationale approach, and improved credibility gained by the IS development team or organisation as a result of a more open design process. Each of these factors may act to support better design decisions and better designs as those closest to the domain of an IS, its end users, could, through access to design rationale, identify areas where the design team had misinterpreted system requirements or were expending resources to meet a requirement of disproportionate importance.

At least one study has called into question the role of participative design in subsequent system acceptance (Mak & Lyytinen, 1997). However, others acknowledge that given the various prostheses developed by end users to support their work with complex tools in an information-rich domain, from a socio-technical perspective all products of IS development are the products of participatory design (Spinuzzi and Zachry, 2000). The successes of the emerging open source software movement suggest that some form of distributed, merit-driven participatory design may lead to better systems, but it is unclear whether the open source model can be adapted within the context of a single organisation.

Some participants argued that exposing the design rationale through an explanation facility was only going half way. They argued that the approach should be extended to support *changing* a design, for example, through configuration switches of a system in production, when the design rationale suggests an improved alternate form. Many desktop software applications support personalisation of the software's behaviour and implementation of most enterprise IS packages involves a complex process of tailoring these application suites to the needs of the organisation. Providing the rationale behind the design of these configuration switches may help users to better understand the implications of configuration settings.

Finally, the DREX approach developed here may be seen to facilitate requirements tracing, assisting the design team as well as end users both in understanding system features relative to their intended scenarios of use and in determining where given design fails to meet the requirements identified early in the systems development lifecycle. Access to design rationale explanations would also help IS users to more fully understand the impact of the inevitable compromises that occur when building a system to meet requirements in an environment of financial, organisational, and other constraints.

The ontological dimension of the research reported here is most closely related to that from Gruber and Russell (1996), who argue for the capture of highly atomic information called "design descriptions" that can be used to generate explanations. The elements of these design descriptions are categorised as follows:

1. Structure – composition of the artefact and relevant topologies
2. Behaviour – what the artefact might do
3. Function – how the artefact achieves its behaviour
4. Requirements – constraints that the artefact must satisfy to perform its behaviour
5. Objectives – desired properties of the artefact that are not directly related to the behaviour such as cost, efficiency, and other factors that may de classified as design criteria

The first three of the categories map to the *what is it?*, *how do I use it?*, and *how does it work?* operational explanations described earlier in this section, while categories 4 and 5 represent a mix of functional and constraint-based explanation content.

However, in the Gruber and Russell work the data required to populate this rich and complex representation schema are assumed to exist in formal engineering models of the device in question. Access to a formal model that includes structural, functional, and causal ordering information represents an idealised explanation knowledge base. While facilitating many possible design-related services including explanation, construction of these models may not be economical for any but the most well-resourced development project where factors such as safety considerations dictate

complete capture of the underlying design specification. The research reported here differs in that the explanations generated use semi-formal models, in the form of QOC-based design rationale, that represent significant economies in terms of the cost of capture. As was discussed in previous chapters, cost of capture represents one of the most significant barriers to adoption of design specification methods, so approaches such as this that help to reduce this cost may inherently increase their viability.

7.1.2 Explanations from Design Rationale

Fred Brooks' classic 1987 paper, *No Silver Bullet: Essence and Accidents in Software Engineering*, (Brooks, 1987) identifies the *invisibility* of software as one of the key problems preventing significant breakthroughs in software development productivity. By invisibility, Brooks is referring to the fact that unlike most other engineering disciplines, construction in software engineering is not bounded by material constraints that make the structure of the artefact designed within these constraints apparent and, in many cases, intuitive (for example, the struts and stays on a suspension bridge). This lack of a visible, tangible structure in response to physical constraints makes the form of an IS architecture in relation to its purpose and environment difficult or impossible to fully comprehend. Attempts to overcome this disconnect between structure and purpose, to make information systems more transparent and thus more intuitive is evident in user interface innovations such as the desktop metaphor and the point-and-click interactions enabled by the mouse.

A fundamental motivation of the research described here is that such metaphorical devices, while representing a significant advance over command-line computing, are only capable of exposing a portion of the underlying structure of a complex IS. Though the approach lacks the subtlety and nuance of a clever, well-implemented user interface metaphor, exposing the system structure behind a user interface via explanations based on its design rationale may be the most literal and economical way to communicate this information. As discussed in Chapter 2, the literature of integrated explanation facilities has suggested a role for design rationale as an explanation content model.

DR Capture

The incremental effort involved in capturing and representing the knowledge required to produce coherent, computer-generated explanations is a significant barrier to the advancement of such technologies. Knowledge capture and representation in a format tractable to both humans and computers necessarily involves the use of formalisms. A plethora of methods for systematising the IS analysis and design process have been proposed over the years, but none have been entirely successful and each has raised a host of issues regarding the most efficient and effective ways to reuse the knowledge and intelligence applied to the systems development task.

In Study 2 reported in Chapter 6, the *Questions, Options, Criteria* (QOC) semi-formal notation (MacLean, et. al., 1996) was employed to structure the design rationale captured retrospectively from the case study design project. QOC was appended in two ways to facilitate access to the design rational knowledge base in response to an IS explanation request. First, an Outline entity was created to map QOC's design *Questions* to the system entities, in this case, object-oriented classes, for which a given question provides explanatory content. This addition proved essential in the process of explanation retrieval and, as importantly, did not detract significantly to the simplicity and ease of use of the QOC notation.

The second amendment was the addition of *type* explanation classifier attributes to both the Question and the *Criterion* QOC elements. The type attribute added to Question is used to classify them according to the explanation-seeking question most closely related (*what is it? how do I use it?, how does it work*). The type attribute added to Criterion is used to classify them as corresponding to the one of the why? explanation content types identified in the DREX framework (e.g., functional explanation, D-N laws, etc.). Though these additions were motivated in large part by research concerns, specifically, analysis of the QOC content relative to the DREX explanation framework, the Question type attribute is also used to order the QOC when delivered as explanation content. The Criterion type may play a role in helping to further select and structure this content in future versions of the Drust tool.

Classifying the design rationale according to these explanation content types did not add significantly to the difficulties in creating the QOC. The most significant overhead in this regard, as suggested in prior research into QOC use (Bellotti, et. al.,

1991; Shum 1991; Buckingham Shum & Hammond, 1994), was the classifying of raw design deliberations into initial and then, especially, into well-formed Questions, Options, and Criteria. Recoding design meetings onto audiotape is relatively un-intrusive and transcribing these recording into full text transcripts may be outsourced and is relatively inexpensive. However, the information generated from this process is voluminous and it requires an individual reasonably skilled in design rationale concepts and the QOC notation, as well as one with in-depth knowledge of the design project under analysis, to construct this record. Though an optimistic perspective on this cost is possible – this translation may be seen facilitating "points of reflection" on the design process and evolving design product (Moran and Carroll, 1996) – the research conducted in this study suggests that a much clearer, positive cost-benefit equation is required in order to justify this activity to project stakeholders.

An important result from the case study reported in Chapter 6 was the extent to which a significant proportion of the overall design rationale appears to elude capture, even in cases such as this where all design meetings and other materials were available for the construction of retrospective rationales. To some extent, this problem could be mitigated by applying process models such as Design Space Analysis (DSA) that are designed to force a more thorough and reflective deliberative structure, though such prescriptions on the process followed by designers often do not meet with full acceptance.

Stefik and Conway (1982) argue that one of the keys to effective knowledge engineering is the promotion of a "common literacy" in the language of system concepts among the members of a practicing community. Their essential point is that such literacy allows members of a community-of-practice to generate, communicate, and reuse knowledge of their activities and that therefore any language proposed for adoption by such a community must fit the modes of practice exhibited by the community if they are to be successfully adopted. It is acknowledged in the KBS literature that a fundamental assumption of the knowledge elicitation process was that one or more domain experts would be available to act as a source for knowledge engineers in a problem domain and that this assumption often proved to be false in the context of actual system development projects subject to organisational resource constraints (Stefik & Conway, 1982). By inculcating a common design 'language' such as DSA/QOC into the practice of a development team, this overhead may be

distributed amongst the entire community potentially resulting in more complete design rationale capture and, through more widespread use, a greater return on the time spent capturing rationale.

Another source of design knowledge that may help to fill these gaps lies in the minds of the individual system developers who translate into working systems both their interpretation of written specifications and, perhaps more importantly, their own understanding of the design model as it evolves through collaborative activities such as design meetings, informal discussions, and e-mail exchanges. In the VentureQuery case study, a demonstrably incomplete design specification nonetheless resulted in a complete working system. Further study is required to determine whether and how the rationale used to bridge these significant gaps can be captured at the individual workstation.

Use cases or scenarios as the source of much of the understanding developed by project team members over the course of a design and development effort represent a relatively untapped resource for explanatory content. The use of scenarios to envision, evaluate, and extend design was pervasive in the VentureQuery case study and the importance of scenarios as the source of design criteria has been highlighted by MacLean and McKerlie (1995). In the study reported here, scenarios contributed to the design rationale only to the extent that they led to the explicit formation of design criteria. Though the analysis and transformation of scenarios would add significantly to the cost of DR, exposing to the user the ways in which concrete examples of potential system use were applied in the design of a system feature may well contribute to the power of a DR-based explanation.

DREX Delivery

The Drust and VentureQuery software prototypes discussed in Chapter 6 demonstrate the feasibility of constructing tools to support user access to explanations based on design rationale. Among the issues that surfaced in their construction and use were explanatory completeness, discussed above, and the difficulty encountered in managing and navigating a design rationale knowledge base of realistic size and scope. The use of a QOC outline to map system concepts to design Questions helped address this issue by providing a top-level entry point into the mass of design rationale and the hypertext navigation mechanism, a natural fit to the directed-graph

structure of QOC 'trees' (Shum, 1991), shifts much of the delivery problem from one of computation to one of information design.

As a representation strategy, hypertext strikes a balance between formal and semi-formal representations. The content of the nodes is generally informal while the structure of the nodes is formal. Though most instructional minimalists maintain that sparse content and the ability to read instructional material in any order without having to work through predefined sequences of potentially irrelevant material is a key tenet of the approach (Carroll, 1990), some concede the practical point that layered approaches to information access such as hypertext can help to resolve problems with the structure of and access to explanatory content detail (Draper, 1998).

Kuwana and Herbsleb (1993) argue that the manner in which information is structured for use in a given domain and for a given application represents a theory about the information itself. They claim that the best of such information 'theories' are those that most effectively capture the salient aspects of the domain or application, are most efficacious in support of communications, and that help to identify incompleteness and inconsistencies in the information structure. In their view, very little research has yet been done to address what kind of structures would come closest to meeting these criteria. As an approach to explanation delivery, the information model implemented in the Drust application is distinctly minimalist in that it is primarily concerned with signposting rather than spoon-feeding elements of an explanatory knowledge base. Much of the computational logic and power is provided by the user of such a content base, the responsibility of the knowledge base designer is to attempt increasingly explorable information structures and, crucially, to ensure that such exploration results in a utility payoff.

According to Minsky (1961, cited in Stefik & Conway, 1982) the key to the information search and search relevance problem is the identification of "planning islands" that help to orient the search, block off dead ends, and suggest the most fruitful avenues for further exploration of the knowledge base. In designing and implementing information systems, we cannot reliably predict the context in which they will be used. Therefore, developing explanation plans that depend on preconceived use scenarios will never provide comprehensive coverage. Borrowing

from Minsky and using the QOC and hypertext models of information structure, the approach taken to explanation delivery here is through the use of *explanation islands* (QOC/hypertext nodes) that help to relate the features and process that make up a system design to the criteria applied in their selection. In effect these explanation islands help to anchor system features or processes to the context from which they emerge. Explanation islands allow users to navigate a base of explanatory knowledge and construct their own explanations from the information provided. In the domain of integrated explanation facilities, an explanation content theory should describe the structure and types of information that are most helpful in orienting a system user's search for understanding, in other words those facts most salient to a useful explanation.

The ideas discussed and investigated in this thesis may be seen as an attempt to develop a content theory to describe such a system of IS explanation islands. Such a theory assists information systems developers by orienting their attempts to identify and apply the criteria that drive their selection of design alternatives. Shum (1991) argues that design criteria 'trees' (multi-level versions of the meta and bridging criteria described by standard QOC) could be used as a mechanism for sensitizing designers to the kinds of issues that are most prevalent (but not necessarily most important) in the design process as well as act as an orienting mechanism for design rationale retrieval. In the VentureQuery case study, for example, the competing meta-criteria of system flexibility and architectural simplicity recur again and again in the design space of individual design decisions. Communicating to end users how designers apply these criteria to ensure the development of both a useful system, and one that is reliable and extensible, may help to explain how an apparently malformed system feature actually represents the a reasonable compromise given constraints.

The VentureQuery design case and resulting prototype software system construction project highlight a number of factors that contribute to understanding of the explanatory content of an IS. One important dimension is the role of the system strategy or system model in helping the design team to understand and organise their approach to the design problem. In the VentureQuery case the system model was that of a board game as metaphor for a questionnaire and this model served to immediately focus the design effort as well as to constrain the options that were identified to address design problems. The use of this organising system model is not readily

apparent in the form of the final system, but by providing access to its effect on design deliberations by access to the design rationale, those not involved in the design process may be able to better understand the "scaffold" upon which many design decisions were made.

7.1.3 Constraints on IS Explanation

The overhead imposed by the use of design rationale techniques to capture detailed system knowledge must be justified within the overall system development cost-benefit equation. If the costs associated with design rationale are justified primarily by the benefits that accrue to the design team (Moran and Carroll, 1996; Conklin & Burgess-Yakemovic, 1996; MacLean, et al., 1996), then the explanation facilities reported here may be seen as yet another benefit, a natural output of system development that minimises the overhead of separate and distinct explanation knowledge acquisition activities. As reported in Chapter 6, design rationale also provides those tasked with producing system documentation, both design specifications and user manuals, with a base of information from which to derive these materials. As reported in Chapter 5, the user documentation phase of a system development project often acts to slow release of an IS or software application since the necessary materials to produce this documentation, including the final form of the system itself, are not available until very near the end of the project. Using the Drust approach described in Chapter 6, design rationale explanations can be captured in parallel with system design evolution and made available immediately to end users and other stakeholders from within the application itself.

Paul (1994) argues that approaches to IS specification and development are broadly conceived as disappointments in that they do not take into account the organic nature of IS needs in rapidly evolving organisations. It is similarly proposed that IS documentation is too frequently looked upon as a singular entity, a "closed system", when in fact it exists in an evolving "ecology" of supplemental information sources such as memos, e-mails, Post-Its, and word-of-mouth (Spinuzzi & Zachry, 2000). As the findings reported here suggest, capturing the full depth of the ecology in which an IS evolves presents significant challenges. Further research is needed into ways in which non-intrusive capture of relevant information from all potential sources can be

supported both from a design process perspective and through the development of automated design capture tools.

7.2 Study Limits

A central goal of the research reported in this dissertation was to begin an exploration into the potential utility of design rationale as a content or knowledge base for explanation facilities that are embedded in an information system. The findings from this exploratory research discussed in this and preceding chapters have generated as many questions as they have addressed, some of these questions will form the basis of future research in this area and are discussed in the section that follows.

The investigation into ideal IS explanations reported in Chapter 5 involved in-depth, semi-structured interviews with IS development professionals. Though the sample represents a broad cross-section of these professionals in different roles, industries, and geographic regions, the number of interviews, 27, represents a relatively small cross-section of what is today a huge, global community of system developers. Generalisations to such a large population from this limited study should be treated with caution. However, one of the goals of the grounded theory approach reviewed and employed in Chapter 5 is to identify a "conceptually dense" set of patterns of action and interaction among actors and processes (Strauss & Corbin, 1994). Though the theoretical framework described here is likely to change as the domain of information systems and their users evolve, it represents an empirical beginning or benchmark from which the framework may also evolve through further study.

The study reported in Chapter 6 was an attempt to investigate a reasonably ecologically valid project in the 'zoo' between the laboratory and the field. Like any single case, the results of such a study may be generalisable to the theory employed (Lee, 1989), but not necessarily to other cases in other settings. The descriptive and/or explanatory utility of a theoretical framework such as the one developed in this work is strengthened as the number of cases and settings to which the theory applies is increased through repeated studies (Yin, 1984).

Both studies employed a mode of data coding, the open and axial coding of grounded theory and the categorisation of design utterances as QOC, that are inherently subjective. This issue is especially acute in the study reported in Chapter 6, as the

difficulty with classifying design issues as Questions, Options, and Criteria are widely acknowledged. A different research with different aims and a different background may well have produced a very different set of results. To address this research dilemma, the methods and techniques used to classify data were thoroughly documented, and chains of evidence (interview transcript -> coded transcripts -> theoretical conjectures, and design meeting transcripts -> QOC with source notations -> theoretical conjectures) exist to show the path from data to reported findings.

As discussed in the section that follows, further studies to investigate the relation of the framework developed here to the explanation content requirements of IS end users over realistic time intervals, i.e., months and years. The results reported here focus primarily on the 'supply side' of the IS explanation equation. More studies are required both in the laboratory and the field to determine the most useful explanation content in the context of use. The framework of explanation content developed here represents only a baseline set of issues that could form the operational core of further work.

Approaches to capturing design rationale must be integrated into the development process in such way as to nearly eliminate the incremental time and effort involved in its capture. Unfortunately, voice recognition, video parsing, and other technologies are not yet mature enough to support zero effort design rationale capture. Until these technologies become viable, approaches that add overhead to the systems development process, such as the one described in Chapter 6, will need to show a clear cost-benefit advantage if they are to be adopted. While this study has begun the work of determining the relative costs and benefits of capturing and delivering design rationale explanations to end users, the results reported here represent suppositions on the part of both study participants and the researcher as to what these potential costs and benefits might be. Additional field studies of IS development projects employing this approach are required to determine the true costs incurred and benefits accrued from the approach described here.

7.3 Topics for Further Research

The research reported in this paper has so far been largely exploratory. Both the research process and the finding have helped to identify a number of areas for further

related research. This section discusses some of the more important areas and suggests ways which future research efforts could help to further elucidate the domain of design rationale explanations.

7.3.1 Explanation Usability

Exploratory, non-experimental studies such as the ones described in this thesis are useful for identifying possible variables for later experimental analysis (Robson, 1994). Some obvious questions related to system usability arise from the analyses presented here. In particular, what do system users, as opposed to developers, think of this explanatory framework? Do they use the explanations and, if so, how do they use them?

An open question is how the perceptions of development professionals reported in Chapter 5 relate to what IS users themselves prefer in terms of explanatory content. Further research is planned to investigate the usability of design rationale explanations in the systems use context. These studies will apply more controlled, experimental and pseudo-experimental methods in contrast to the largely qualitative methods employed in this project to date. Among the questions this research will attempt to address are:

1. Does access to design rationale information and design rationale-based explanations contribute to the usability of a software information system?
2. What types of explanatory content are preferred by IS users and what types actually contribute to their performance?
3. Do system users access design rationale information when it is available?
4. Are particular components of the design rationale more important to the usability equation?
5. Are different types of systems, e.g., systems in different domains or at different levels of complexity, made more or less usable given user access to design rationale?

The study of human-computer interface issues has become increasingly important in the fields of computing and information systems and integrated explanation facilities

have been identified as a crucial component of the interface to knowledge-based systems (Lamberti & Wallace, 1990). Prior research suggests that 50% or more of a modern application's source code may be dedicated to supporting the human-interface (Marcus and van Dam, 1991), making this a significant proportion of the system design content. Examples of the importance of the human-interface to the overall success of an information system are common. In a comparative study of the difficulties encountered learning mathematics versus learning a computer application, a text processor in this case, Lewis (1988) found that the computer and its human interface are the source of significant additional problems in the learning task. He argues that the complexity of the human interface is the decisive factor in how difficult a computer program is to learn and to use. Because errors are compounded, one error or erroneous assumption leads to further errors, even apparently trivial interface details must be analysed for their impact on usability. Explanations may play a central role in the achievement of quality human-interface design.

In their early work on the psychology of human-computer interaction, Card, et al. (1983) enumerated a set of performance variables that can be used to measure the effectiveness of computer system's user interface. The first is *functionality*, what is it that the system can do? Another is *learning*, how long does it take for a user to acquire functional capabilities? Next is *time*, how long does it take to accomplish a given task with the system? *Quality* is a crucial factor, how frequently do user errors occur, what are their consequences, and to what standard are the computer-supported tasks performed? Two other variables address issues of human performance, in particular, the performance demands that are made on the user; *fatigue* and *stress* and *working* and *long-term memory*. The final criterion that they outline is *subjective*, users' opinions of the system. Murphy, et al. (1999) characterise research into software technologies as centrally concerned with both the usefulness and the usability of any new approach. This dichotomy is echoed in studies of technology adoption in the MIS context, where the widely utilised Technology Acceptance Model (TAM) is used to investigate the interplay of usefulness and ease of use as factors central to the adoption of any new IS technology (Davis, 1989).

In this thesis it is argued that the explanatory knowledge underlying an IS is only partially embodied in the system artefact, and that significant additional understanding may be derived from the design rationale behind the artefact. This is

only one part of the IS explanation picture. When a system is complete but not yet implemented, the artefact and a comprehensive design rationale may constitute all of the information needed to explain and understand the system. However, once the system is in production, out in the 'world' with real users and a complex web of organisational and social factors contributing to the use context, this situation changes. Accounting for this change to any degree, if it is practically possible at all, would require additional mechanisms.

Excluded from the research reported here is the role of multimedia in explanation. Work such as that of Tufte (1983; 1990; 1997) points to the importance of graphics in visual explanation. The idea that graphics, sounds, animations and other multimedia devices can assist in the explanation task by retaining the interest of the explanation recipient is often expounded (Woolf, 1996). Given the pervasiveness of "box and arrow" type formalisms in the IS design process, inclusion of these types of graphics in explanations seems an obvious augmentation to the text-only explanation provided by the software tools developed as part of this study. Applying Tufte's ideas to the construction of explanations that make use of diagrams and other graphic representations presents interesting opportunities and challenges for the design rationale-based view of IS explanation.

7.3.2 Tailored DR Explanations

Given the apparent importance of the pragmatic explanations identified in Chapter 5, a central problem in explanation delivery is finding the balance between the needs of different users in different task scenarios. Prior research has identified the differences between novice and experts users as particularly important (Patrick & McGurgan, 1993). It has been proposed that some knowledge of the receiver of design rationale information, for example, whether novice or expert, could assist with tailoring the information provided (Ramesh & Sengupta, 1995). For example, some research suggests that more sophisticated users are more likely to prefer information regarding the underlying rationale of a system, in particular, the way in which the design of the system was related to tasks in the problem domain (Hackos, 1998).

The use of computer-generated natural language and user modelling for the communication of system generated explanations is controversial. Attempts at computer-generated instructional dialogues often appear contrived to their human

recipients. (Wenger, 1987) and Zimmer (1989) claims that in many cases natural language systems have been rejected by users because they tend to feel 'tricked' by protracted and misleading 'natural' responses. Zimmer claims that the shallowness of these surface natural language systems suggest that we should not concentrate on imitating language pragmatics but on integrating these pragmatics into the specification of the system.

Among the alternatives to user-modelling and explanation dialogue management that may be applied to modulating the explanation content provided by an IS is the use of hypertext, as applied in the research reported here, where continuous expansion of the knowledge underlying the rule is made possible. Further empirical testing of the utility of different explanation content could lead to the development of better "explanation islands", which help to guide users as they expand the explanation in particular directions. The nature of these islands might include level of detail of the explanation, or whether D-N constraints or functional explanatory content is needed.

Future research may also investigate whether certain types of explanation content are more useful in certain domains. For example, explanation facilities may be especially important in so-called vertical IS applications where the system is domain knowledge rich and is targeted to expert users.

7.3.3 Longitudinal Studies of Design Rationale Explanations

The kinds of deep explanation provided by design rationale may resist experimentation without the ecological validity provided by longitudinal study. An experimental subject working through a timed IS problem most likely would not be able to appreciate the level of detail and consequent understanding provided by design rationale explanations. It may be that the actual utility of such explanations can only be observed and measured over extended periods of time in the context of working systems use in a valid organisational context. Such studies are difficult to initiate, manage, and complete, and studying software evolution over realistic time periods is even more difficult given the lack of theory, models, and especially the organisational access that is required to track system development and evolution (Kemerer & Slaughter, 1999). However, especially when they are available to be analysed along with the results of both exploratory and experimental research, the results of

longitudinal studies represent the arbiter of whether user interface innovations enhance the experience of a given system's constituency.

7.3.4 Mining Design Rationale

Given the mass of information that can result from the collection of design rationale for a realistically-sized development project, an important area for future research is into the technologies that could be developed to support "mining" this mass for its most valuable contributions to IS understanding. In particular, technologies that support mining design rationale data for the explanatory relationships that may exist between different components may result in the discovery of new information for explaining a design. According to Dreyfus and Dreyfus (1990), many human-to-human explanations rely on the ability of the person doing the explaining to use unrelated knowledge and common sense drawn from outside the realm of the domain in question. Analogy, metaphor, and comparison are complex linguistic constructs that are frequently used in everyday explanation and these present intractable problems with today's explanation technology. Our lack of understanding of the issues involved in acquisition, representation and storage, and expression of human-like explanation knowledge must be addressed before truly expert explanation systems can be developed.

Among the technologies that might help to address this issue, while still retaining the essential quality of supporting active, exploratory learning on the part of the user are advanced information retrieval and information filtering. Belkin and Croft (1992) define information retrieval systems as those that include technologies designed to lead people quickly to the information that they need. They define information filtering systems as those that help people to make more effective and efficient use of available information sources including large, relatively unstructured data sets. Information filtering systems are those that include consideration of information usage patterns, the relevance and value of information to a particular user over time, and the content attributes that make some types of information more valuable than others (Loeb, 1992).

The widespread adoption of the Internet and World Wide Web has resulted in both increased research interest and commercialisation of sophisticated information retrieval and filtering technologies. Many of these have the potential to facilitate

improved access to knowledge bases such as those represented by design rationale. A promising example of these technologies is the inverted index approach, where an index of the words that occur in a set of documents is constructed with pointers to the documents in which a given word appears (Dreilinger & Howe, 1997). The index may be augmented with information about the number of occurrences of a given word within a given document and with the exact location of a word within a document. Though this additional information increases the size of the index, it also facilitates more complex search techniques and faster searches. Searches that make use of this method consider term frequency, the number of times that a term occurs in a document, and inverse document frequency, terms that appear in fewer documents are given additional relevance weights, in selecting and rank ordering the results of the search. Techniques such as this applied to the filtering and retrieval of design rationale explanations may help to mitigate difficulties encountered in locating the most appropriate information in a given IS use context.

7.4 Conclusion

As a topic of research, the problems of IS explanations, integrated explanation facilities, and computer-generated explanation may be conceptualised from several different perspectives. One such perspective is the computational approach, where explanation is viewed within the artificial intelligence (AI) domain and which, like many AI problems, is reducible to issues of knowledge representation and search. Another perspective is what may be termed the relativist approach, which views the nature of explanation as entirely interest and context dependent, and which therefore suggests that without the singular, scenario-dependent knowledge possessed by the explanation requestor, any attempt to pre-determine a system user's explanation needs is an invitation to failure. Though prior research in the computational tradition has attempted to address the issue of interest-relative explanation through sophisticated systems of user models and explanatory dialogue management systems, to date these systems are not widely accepted.

The approach developed in this dissertation provides yet another perspective on the problem of IS explanation. As the results of the study described in Chapter 5 suggest, the content of an effective IS explanation may indeed depend on the relative interests of the explanation requestor. However, rather than relying on a program or algorithm

to determine these interests, the design presented in Chapter 6 explicitly eschews the goals of user modelling and computer dialogue management in favour of a content-centric approach that promotes the construction of explanations *by the system user* through active, exploratory learning. This approach makes use of mature, proven, and inexpensive technologies (e.g., Java, a DBMS, and hypertext) and shifts the problem focus to areas of explanation content identification, explanation content sourcing and capture, and explanation information design.

An essential problem with the understanding and use of software information systems is the invisibility of the underlying structure of these systems relative to the functions that they perform (Brooks, 1987). Software designs consist of systems of abstractions, and the received view in software engineering practice is that the more abstract a software component, the more flexible the component becomes in terms of the number of use scenarios the component supports. This leads to a problem. The users of a system do not necessarily understand the process by which these abstractions are derived from the myriad concrete contexts that they do understand, and that make up the domain in which the system is intended to be used. These scenarios of use (Carroll, 1995), act as a starting point for system design, for example, in the form of use case diagrams in the UML, but as analysis, design, and construction progresses the system design that evolves to realize these requirements is both moulded and distorted by a diverse array of factors including the material constraints of hardware and software performance, the psychological constraints of cognition, organizational constraints of time and other resources, and social constraints imposed by standards bodies and accepted professional best practice. This disjunction, Norman (1986) has called it the "gulf of understanding", that exists between a system's (really, the designer's) understanding of the use context and the user's understanding of how the system fits this context is the target of the approach reported in this dissertation.

An enduring dream for AI researchers is that the products of their research will someday form an important component of the information society, supporting "knowledge workers" as they capture and reuse intellectual capital towards the goals of organisations and society (Hayes-Roth & Jacobstein, 1994). By providing system design information a form amenable to explanation construction, the DREX framework and the Drust protype make a contribution to what has been called the "ecology" of system use (Spinuzzi & Zachry, 2000). A problem with explanation in

IS work to date is lack of a sound theoretical base to orient current and future research efforts (Gregor & Benbasat, 1999). Of particular interest to the research reported here is that little work on IS explanation has focused on the content of system explanation and how that content might be obtained. These questions are crucial to practitioners; frameworks that help them decide what sort of information is most likely to help stakeholders understand their systems are a first step towards explanation. Defined approaches to obtaining this content along with case studies describing the approach in action help clarify the true costs and other overheads associated with these methods.

References

Achinstein, P. (1983). *The Nature of Explanation.* New York: Oxford University Press.

Albert, S. & Bradley, K. (1997). *Managing Knowledge: Experts, Agencies and Organizations*, Cambridge, UK: Cambridge University Press.

Alexander, C. (1964). *Notes on the Synthesis of Form.* Cambridge, MA: Harvard University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language.* Oxford, UK: Oxford University Press.

Al-Jumeily, D. & Strickland, P. (1997). Designing an Interface on the Web for an Intelligent Tutoring System. *Proceedings of the 23rd EUROMICRO conference* (pp. 158-162).
Available at: www.cms.livjm.ac.uk/research/aai/publications.htm#1997

Allen, R. B. (1990). User models: theory, method, and practice. *Int'l. J. Man-Machine Studies, 32*, 511-543.

Angelides, M. C. & Paul, R. J. (1993). Towards a Framework for Integrating Intelligent Tutoring Systems and Gaming Simulation. In G. W. Evans, M. Mollaghasemi, E. C. Russell, W. E. Biles, (Eds.), *Proceedings of the Winter Simulation Conference* (pp. 1281-1289). New York: ACM.

Antaki, C. (Ed.). (1988). *Analysing Everyday Explanation: A Casebook of Methods.* London: Sage.

Antaki, C. (1989). Lay explanations of behaviour: how people represent and communicate their ordinary theories. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 201-212). Chichester, UK: Ellis Horwood.

Antaki, C. (1994). *Explaining and Arguing: The Social Organization of Accounts.* London: Sage.

Avgerou C. & Cornford T. (1998). *Developing information systems: concepts, issues and practice* (2nd ed.). London: Macmillan.

Avison, D. E. & Fitzgerald, G. (1988). *Information Systems Development: Methodologies, Techniques and Tools.* London: Blackwell Scientific Publications.

Avison D. E. & Fitzgerald G. (1995). *Information systems development: methodologies, techniques and tools* (2nd ed.). London: McGraw-Hill.

Baecker, R. M. (Ed.). (1993). *Readings in groupware and computer-supported cooperative work: assisting human-human collaboration.* San Mateo, CA: Morgan Kaufmann Publishers.

Bainbridge, B. (1988). The Explicit Representation of Control Knowledge. In G. A. Ringland & D. A. Duce (Eds.), *Approaches to Knowledge Representation - An Introduction* (pp. 175-187). Taunton, England: Research Studies Press. Available at: www.dfki.uni-sb.de/imedia/lidos/bibtex/Ringland_a26288-98.html

Barber, P. (1988). *Applied Cognitive Psychology - An information-processing framework.* London: Methuen.

Basili, V. R., Shull, F. & Lanubile, F. (1999). Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering, 25*(4), 456-473.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Boston: Addison-Wesley.

Belkin, N. J. & Croft, W. B. (1992). Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM,* 35(12), 29-38.

Bellotti, V. (1993). Integrating Theoreticians' and Practitioners' Perspectives with Design Rationale. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'93)* (pp. 101-114). New York: ACM.

Bellotti, V., MacLean, A., & Moran, T. (1991). *Generating Good Design Questions.* Available as technical report EPC-1991-136, Rank Xerox Research Centre, Cambridge Laboratory.

Benbasat, I. & Zmud, R. W. (1999). Empirical Research in Information Systems: The Practice of Relevance. *MIS Quarterly, 23*(1), 3-16.

Benbasat, I. & Weber, R. (1996). Research Commentary: Rethinking Diversity in Information Systems Research. *Information Systems Research, 7*(4), 389-399.

Berg, B. L. (1998). *Qualitative Research Methods for the Social Sciences (3rd ed.)* Boston: Allyn and Bacon.

Berry, D. C. & Broadbent, D. E. (1987). Expert systems and the man-machine interface. Part Two: The user interface. *Expert Systems, 4*(1), 18-27.

Bidgoli, H. (1996). Group Support Systems: A New Productivity Tool for the 90's. *Journal of Systems Management, 47*(4), 56-62.

Bird, A. (1998). *Philosophy of Science.* London: UCL Press.

Block, N. J. (Ed.). (1980). *Readings In Philosophy Of Psychology*. Cambridge, MA: Harvard University Press.

Boden, M. A. (1990). Introduction. In M. A. Boden (Ed.), (pp. 1-21). Oxford, UK: Oxford University Press.

Boehm, B. W. (1981). *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. *Computer, 21*(5), 61-72.

Bose, P. (1998). Change Analysis in an Architectural Model: A Design Rationale Based Approach. *Proceedings of the 3rd international workshop on software architecture.* New York: ACM.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications* (2nd ed.). Reading, MA: Addison-Wesley.

Booch, G., Jacobson, I., Rumbaugh, J. (1998). *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.

Bowker, G. & Star, S. L. (1994). Knowledge and infrastructure in international information management: Problems of classification and coding. In L. Bud-Frierman (Ed.), *Information Acumen: The understanding and use of knowledge in modern business* (pp. 187-216). London: Routledge.

Bradshaw, J. A. & Young, R. M. (1991). Evaluating Design Using Knowledge of Purpose and Knowledge of Structure. *IEEE Expert, 6*(2), 33-40.

Brockmann, R. J. (1990). The Why, Where and How of Minimalism. *Proceedings of the conference on SIGDOC '90* (pp. 111-119). New York: ACM.

Brooks, F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer, 20*(4), 10-19.

Brown, J. S. (1984). The Low Road, the Middle Road, and the High Road. In P. A. Winston & K. A. Pendergast (Eds.), *The AI Business: Commercial Uses of Artificial Intelligence* (pp. 81-90). Cambridge, MA: MIT Press.

Browne, D. (1990). Conclusions. In D. Browne, P. Totterdell, & M. Norman (Eds.), *Adaptive User Interfaces* (pp. 195-222). London: Academic Press.

Brunswik, E. (1952). *The Conceptual Framework of Psychology*. Chicago: University of Chicago Press.

Buchanan, B. G. & Shortliffe, E. H. (Eds.). (1984). *Rule-Based Expert Systems.* Reading, MA: Addison-Wesley.

Buckingham Shum, S. & Hammond, N. (1994). Argumentation-Based Design Rationale: What Use at What Cost? *International Journal of Human-Computer Studies, 40*(4), 603-652.

Buckingham Shum, S., MacLean, A., Bellotti, V. & Hammond, N. (1997). Graphical Argumentation and Design Cognition. *Human-Computer Interaction, 12*(3), 267-300.

Buckingham Shum, S. (1996). Analyzing the Usability of a Design Rationale Notation. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 185-216). Mahwah, NJ: Lawrence Erlbaum.

Buckingham Shum, S. (2000). Computer-Supported Collaborative Argumentation Resource Site, The Knowledge Media Institute, The Open University. Available at: kmi.open.ac.uk/people/sbs/csca/index.html

Buckland, J. A., Fowinkle, R., Shroyer, L., & Rice, V. (1991). *Total Quality Management in Information Systems.* Boston: QED Technical Publishing Group.

Bush, V. (1945, July). As We May Think. *The Atlantic Monthly, 176*(1), 101-108.

Byrne, R. W. (1989). Social expertise and verbal explanation: the diagnosis of intelligence from behaviour. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 76-91). Chichester, UK: Ellis Horwood.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Carey, T., McKerlie, D., & Wilson, J. (1996). HCI Design Rationale as a Learning Resource. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 21-51). Mahwah, NJ: Lawrence Erlbaum.

Carroll, J. M. (1987). Preface. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction.* Cambridge, MA: MIT Press.

Carroll, J. M. (1990). *The Nurnberg Funnel: Designing minimalist instruction for practical computer skill.* Cambridge, MA: MIT Press.

Carroll, J. M. (ed.) (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development.* New York: John Wiley.

Carroll, J. M. (1997). Human-computer interaction: psychology as a science of design. *International Journal of Human-Computer Studies, 46,* 501-522.

Carroll, J. M. (1998). Reconstructing Minimalism. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp.1-17). Cambridge, MA: MIT Press.

Carroll, J. M. & Aaronson, A. P. (1988). Learning by Doing with Simulated Intelligent Help. *Communications of the ACM, 31*(9), 1064-1079.

Carroll, J. M., Alpert, S. R., Karat, J., Van Deusen, M., Rosson, M. B. (1994). Raison d'Etre: Capturing Design History and Rationale in Multimedia Narratives, Celebrating Independence. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)* (pp. 192-197). New York: ACM.

Carroll, J. M., Mack, R. L., Robertson, S. P. & Rosson, M. B. (1994). Binding objects to scenarios of use. *International Journal of Human-Computer Studies*, 41, 243-276.

Carroll, J. M. & Rosson, M. B. (1987). Paradox of the Active User. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 80-111). Cambridge, MA: MIT Press.

Carroll, J. M. & Rosson, M. B. (1996) Deliberated Evolution: Stalking the View Matcher in Design Space. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 107-145). Mahwah, NJ: Lawrence Erlbaum.

Cawsey, A. (1993). *Explanation and Interaction: The Computer Generation of Explanatory Dialogues*. Cambridge, MA: MIT Press.

Chamberland, L. (1999). Componentization of HTML-Based Online Help. *Proceedings of the seventeenth annual international conference on Computer documentation*, October 1999, 165-168.

Chandrasekaran, B. & Mittal, S. (1983). Deep versus compiled knowledge approaches to diagnostic problem-solving. *Int'l. J. Man-Machine Studies, 19*, 425-436.

Chandrasekaran, B. & Swartout, W. (1991). Explanations in Knowledge Systems. *IEEE Expert, 6*(3), 47-49.

Chandrasekaran, B., Josephson, J. R., & Benjamins, V. R. (1999). What Are Ontologies and Why Do We Need Them? *IEEE Intelligent Systems, 14*(1), 20-26.

Checkland P. (1981). *Systems thinking, systems practice.* Chichester: John Wiley.

Churchland, P. M. (1989). *A Neurocomputational Perspective: The Nature of Mind and the Structure of Science.* Cambridge, MA: MIT Press.

Clancey, W. J. & Letsinger, R. (1981). NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (pp. 829-836). Los Altos, CA: Morgan Kaufmann.

Clancey, W. J. (1983). The Epistemology of a Rule-Based Expert System - A Framework for Explanation. *Artificial Intelligence, 20*(3), 215-251.

Clancey, W. J. (1986, August). From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985. *AI Magazine, 7*(3), 40-60.

Clancey, W. J. (1993a). Notes on "Epistemology of a Rule-based Expert System". *Artificial Intelligence, 59*(1-2), 191-204.

Clancey, W. J. (1993b). Guidon-Manage Revisited: A Socio-Technical Systems Approach. *Journal of Artificial Intelligence in Education, 4*(1), 5-34.

Cleal, D. M. & Heaton, N. O. (1988). *Knowledge-Based Systems - implications for human-computer interaction*. Chichester, UK: Ellis Horwood.

Conklin, E. J. & Burgess-Yakemovic, K. C. (1996). A Process-Oriented Approach to Design Rationale. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 393-427). Mahwah, NJ: Lawrence Erlbaum.

Conklin, J. & Begeman, M. L. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems, 6*(4), 303-331.

Cooke, N. J. (1989). The elicitation of domain-related ideas: stage one of the knowledge acquisition process. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 58-75). Chichester, UK: Ellis Horwood.

Cross, N. (Ed.). (1984). *Developments in Design Methodology*. Chichester, UK: John Wiley.

Cummins, R. (1975). Functional Analysis. *The Journal of Philosophy, 72*, 741-765.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM, 31*(11), 1258-1287.

Cusumano, M. A. & Selby, R. W. (1997). How Microsoft Builds Software. *Communications of the ACM, 40*(6), 53-61.

Davidson, D. (1963). Actions, Reasons and Causes. In D. Davidson (Ed.), *Essays on Actions and Events* (pp. xx-xx). Oxford, UK: Oxford University Press. Reprinted in M. Martin & L. C. McIntyre, L.C. (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 675-686). Cambridge, MA: MIT Press.

Davidson, D. (1974). Psychology as Philosophy. In S.C. Brown (Ed.), *Philosophy of Psychology* (pp. 41-52). London: Macmillan Press. Reprinted in M. Martin & L. C.

McIntyre, L.C. (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 79-89). Cambridge, MA: MIT Press.

Davis, F.D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly, 13*(3). 319-339.

Davis, R. (1984). Amplifying Expertise with Expert Systems. In P. H. Winston (Ed.), *The AI Business: The Commercial Uses of Artificial Intelligence*. Cambridge, MA: MIT Press.

Dawkins, R. (1989). *The Selfish Gene, New Edition*. New York: Oxford University Press.

DeMarco, T. (1979). *Structured analysis and systems specification.* Englewood Cliffs, NJ: Prentice-Hall.

Dennett, D. C. (1987). *The Intentional Stance*. Cambridge, MA: MIT Press/Bradford Books.

Dennett, D. C. (1990). Cognitive Wheels: The Frame Problem of AI. In M. A. Boden (Ed.), *The Philosophy of Artificial Intelligence*. Oxford, UK: Oxford University Press.

Denning, P. (1982). Electronic Junk. *Communications of the ACM, 25*(3), 163-165.

Devanbu, P., Brachman, R. J., Selfridge, P. G., & Ballard, B. W. (1991). LaSSIE: A Knowledge-based Software Information System. *Communications of the ACM, 34*(5), 34-49.

Dhaliwal, J. S. & Benbasat, I. (1996). The Use and Effects of Knowledge-based System Explanations: Theoretical Foundations and a Framework for Empirical Evaluation. *Information Systems Research, 7*(3), 342-362.

Dhaliwal, J. S. & Tung, L. L. (2000). Using group support systems for developing a knowledge-based explanation facility. *International Journal of Information Management, 20*, 131-149.

Dore, R. P. (1961). Function and Cause. *American Sociological Review, 16*, 843-853. Reprinted in M. Martin & L. C. McIntyre (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 377-389). Cambridge, MA: MIT Press.

Draper, S. W. (1987). Spontaneous Explanation. *Proceedings of the 3rd ALVEY Explanation Conference*, 74-82.

Draper, S. W. (1988). What's going on in everyday explanation? In C. Antaki (Ed.), *Analysing Everyday Explanation* (pp. 15-31). London: Sage Publications.

Draper, S. W. (1998). Practical Problems and Proposed Solutions in Designing Action-Centered Documentation. In J. M Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 349-374). Cambridge, MA: MIT Press.

Dreilinger, D. & Howe, A. E. (1997). Experiences with Selecting Search Engines Using Metasearch. *ACM Transactions on Information Systems, 15*(3), 195-222.

Dreyfus, H. L. (1991). *What Computers Still Can't Do.* Cambridge, MA: MIT Press.

Dreyfus, H. L., & Dreyfus, S. E. (1986). *Mind Over Machine.* Oxford, UK: Basil Blackwell.

Dreyfus. H. L., & Dreyfus, S. E. (1990). Making a Mind Versus Modelling the Brain: Artificial Intelligence Back at a Branch-Point. In M. A. Boden (Ed.), *The Philosophy of Artificial Intelligence.* Oxford, UK: Oxford University Press.

Eberhart, R. C. (1995). Using Evolutionary Computation Tools in Explanation Facilities. *International Journal of Expert Systems, 8*(3), 277-285.

Edwards, A. & Connell, N. A. D. (1989). *Expert Systems in Accounting.* Hemel-Hempstead, UK: Prentice-Hall.

Elam, J. J., Walz, D. B., Curtis, B. & Krasner, H. (1991). Measuring Group Process in Software Design Teams. In H.-E. Nissen, H. K. Klein, & R. Hirscheim (Eds.), *Information Systems Research: Contemporary Approaches and Emergent Traditions* (pp. 51-61). Amsterdam: North-Holland.

Ellis, C (1989). Explanation in intelligent systems. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 108-126). Chichester, UK: Ellis Horwood.

Elsbach, K. D. & Elofson, G. (2000). How the Packaging of Decision Explanations Affects Perceptions of Trustworthiness. *Academy of Management Journal, 43*(1), 80-89.

Elster, J. (1985). The Nature and Scope of Rational-Choice Explanation. In E. LePore and B. McLaughlin (Eds.), *Actions and Events: Perspectives on Donald Davidson* (pp. 60-72). Oxford, UK: Blackwell. Reprinted as Functional Explanation: In Social Science in M. Martin & L. C. McIntyre (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 403-414). Cambridge, MA: MIT Press.

Engelbart, D. C. (1963). A Conceptual Framework for the Augmentation of Man's Intellect. In P. W. Howerton & D. C. Weeks (Eds.), *Vistas in Information Handling, vol. 1*, (pp. 1-29). Washington: Spartan Books. Reprinted in Mayer, P.

A. (Ed.), (1999), *Computer Media and Communication: A Reader* (pp. 72-96). Oxford, UK: Oxford University Press.

Farkas, D. K. & Williams, T. R. (1990). John Carroll's *The Nurnberg Funnel* and Minimalist Documentation (book review and critique). *IEEE Transactions on Professional* Communication, *33*(4), 182-187.

Fischer, G. (1999). Domain-Oriented Design Environments: Supporting Individual and Social Creativity. In J. Gero and M. L. Maher (Eds.), *Computational Models of Creative Design IV*, 83-111. Key Centre of Design Computing and Cognition, Sydney, Australia.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. L. (1996) Making Argumentation Serve Design. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 267-293). Mahwah, NJ: Lawrence Erlbaum.

Flowers, S. (1996). *Software Failure, Management Failure - Amazing Stories and Cautionary Tales*. Chichester, UK: John Wiley.

Foltz, P. W. & Dumais, S.T. (1992). Personalized Information Delivery: An Analysis of Information Filtering Methods. *Communications of the ACM, 35*(12), 51-60.

Fontana, A. & Frey, J. H. (1994). Interviewing: the Art of Science. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research* (pp.361-376). London: Sage Publications.

Forsythe, D.E. (1997). Representing the User in Software Design. Unpublished manuscript available at: www.stanford.edu/dept/HPS/forsythe.paper.html.

Fowler, M. and K. Scott (2000). *UML Distilled* (2nd ed.). Reading, MA: Addison-Wesley.

Frankfort-Nachmias, C. & Nachmias, D. (1996). *Research Methods in the Social Sciences* (5th ed.). London: Arnold.

Frechtling, J. & Sharp, L. (1997). Introducing This Handbook. In J. Frechtling & L. Sharp (Eds.), *User-Friendly Handbook for mixed Method Evaluations, NSF97-153*. National Science Foundation.
Available at: www.her.nsf.gov/HER/REC/pubs/NSF97-153

Freedman, R., Syed, S. A., & McRoy, S. (2000). Links: what is an intelligent tutoring system? *Intelligence, 11*(3), 15-16.

Freeman , L. E. (2000). The supply and demand of information systems doctorates: Past, present, and future. *MIS Quarterly, 24*(3), 355-381.

Friedman, M. (1988). Explanation and Scientific Understanding. In J. Pitt (Ed.), *Theories of Explanation* (pp. 188-198). Oxford, UK: Oxford University Press.

Galliers, R. D. & Land, F. F. (1987). Choosing Appropriate Information Systems Research Methodologies. *Communications of the ACM, 30* (11), 900-902.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

Garnham, A. (1988). *Artificial Intelligence An Introduction.* London: Routledge & Kegan Paul.

Gasper, P. (1991). Causation and Explanation. In R. Boyd, P. Gasper, & J. D. Trout (Eds.), *The Philosophy of Science.* Cambridge, MA: MIT Press.

Gibbs, S., Tsichritzis, D., Casais, E. (1990). Class Management for Software Communities. *Communications of the ACM, 33,* (9), 90-103.

Gill, T. G. (1995). Early Expert Systems: Where Are They Now? *MIS Quarterly 19*(1), 51-81.

Gill, T. G. (1996). Expert Systems Usage: Task Change and Intrinsic Motivation. *MIS Quarterly, 20*(3), 301-329.

Glaser, B. & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research.* Chicago: Aldine.

Glass, R. L. (1996). The Relationship Between Theory and Practice in Software Engineering. *Communications of the ACM, 39*(11), 11-13.

Glass, R. L. (1997). *Software Runaways.* Upper Saddle River, NJ: Prentice-Hall.

Goldstein, N. & Alger, J. (1992). *Developing Object-Oriented Software for the Macintosh: Analysis, Design, and Programming.* Reading, MA: Addison-Wesley.

Gopnik, A., Glymour, C., & Sobel, D. (1999). Causal maps and Bayes nets: A cognitive and computational account of theory formation. *Proceedings of the 11th International Congress on Logic, Methodology, and Philosophy of Science,* Cracow, Poland.
Available at: www-psych.stanford.edu/~jbt/224/Gopnik_1.html

Gotel, O. & Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem. *Proceedings of the First International Conference on Requirements Engineering* (pp.94-101).

Gould, J. D. & Lewis, C. (1985). Designing for usability: Key Principles and What Designers Think. *Communications of the ACM, 28*(3), 300-311.

Greenbaum, J. & Kyng, M. (Eds.) (1991). *Design at Work: Cooperative Design of Computer Systems*. London: Lawrence Erlbaum.

Gregor, S. & Benbasat, I. (1999). Explanations from Intelligent Systems: Theoretical Foundations and Implications for Practice. *MIS Quarterly, 23*(4), 497-530.

Grice, R.A. (1989). Online Information: What Do People Want? What Do People Need? In *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, E. Barrett, (Ed.), Cambridge, MA: MIT Press.

Gruber, T. (1991). Learning Why by Being Told What. *IEEE Expert, 6*(4), 65-74.

Gruber, T. R. & Russell, D. M. (1996). Generative Design Rationale: Beyond the Record and Relay Paradigm. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 21-51). Mahwah, NJ: Lawrence Erlbaum.

Grudin, J. (1991). CSCW (Introduction to special issue). *Communications of the ACM, 34*(12), 30-34.

Grudin, J. (1994). Computer-Supported Cooperative Work: History and Focus. *IEEE Computer, 27*(5), 19-26.

Guindon, R. (1990). Knowledge exploited by experts during software system design. *Int. J. Man-Machine Studies, 33*, 279-304.

Hackos, J. T. (1998). Choosing a Minimalist Approach for Expert Users In J. M Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp.149-177). Cambridge, MA: MIT Press.

Hair, D. C. & Lewis, C. (1990). *Are Argument Representation Schemes Useful?* Available as technical report CU-CS-475-90, The University of Colorado at Boulder, Department of Computer Science.

Hakim, C. (1987). *Research Design: Strategies and Choices in the Design of Social Research*. London: Routledge.

Hannan, M. T. & Freeman, J. (1989). *Organizational Ecology*. Cambridge, MA: Harvard University Press.

Harandi, M. T. (1988) . Building a Knowledge-Based Software Development Environment. *IEEE Journal on Selected Areas in Communications, 6* (5), 862-868).

Harman, G. (1986). *Change in View: Principles of Reasoning*. Cambridge, MA: MIT Press/Bradford Books.

Hasling, D. W., Clancey, W. J. & Rennels, G. (1984). Strategic explanations for a diagnostic consultation system. *Int'l. J. Man-Machine Studies, 20*, 3-19.

Hayes, N. (1997). Theory-led thematic analysis: Social identification in small companies. In N. Hayes (ed.) *Doing Qualitative Analysis in Psychology* (pp.93-114). Hove, U.K.: Psychology Press.

Hayes-Roth, F. & Jacobstein, N. (1994). The State of Knowledge-Based Systems. *Communications of the ACM, 37*(3), 27-39.

Hayes-Roth, F., Waterman, D. A., & Lenat, D. (1983). *Building Expert Systems.* Reading, MA: Addison-Wesley.

Haynes, S. R. (2000). Explanation in Information Systems: Can Philosophy Help? *Proceedings of the 8th European Conference on Information Systems (ECIS 2000)* (pages 31-38). Vienna, Austria.

Heckerman, D., Breese, J. S., & Rommelse, K. (1995). Decision-Theoretic Troubleshooting. *Communications of the ACM, 38*(3), 49-57.

Hedberg, S. R. (1998). Is AI going mainstream at last? A look inside Microsoft Research. *IEEE Intelligent Systems*, March/April 1998, 21-25.

Heidegger, M. (1967/1927). *Being and Time.* Oxford, UK: Blackwell.

Hempel, C. G. (1942). The Function of General Laws in History. *Journal of Philosophy, (39),* 35-48, Reprinted in M. Martin & L. C. McIntyre (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 43-53). Cambridge, MA: MIT Press.

Hempel, C.G. (1965). Aspects of Scientific Explanation. In *Aspects of Scientific Explanation and Other Essays.* New York: Free Press.

Hempel, C. G. & Oppenheim, P. (1948). Studies in the Logic of Explanation. In J. Pitt (Ed.), *Theories of Explanation.* Oxford, UK: Oxford University Press.

Herbsleb, J. D. & Kuwana, E. (1993). Preserving Knowledge in Design Projects: What Designers Need to Know. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'93)* (pp. 7-14). New York: ACM.

Heylighen, F. (1992). Principles of Systems and Cybernetics: an evolutionary perspective. In R. Trappl (Ed.), *Cybernetics and Systems '92* (pp. 3-10). Singapore: World Science.

Higa, K., Morrison, M., Morrison, J., Sheng, O.R.L. (1992). An Object-Oriented Methodology for Knowledge Base/Database Coupling. *Communications of the ACM, 35*(6), 99-113.

Hill, A. B. (1965). The environment and disease: association or causation? *Proceedings of the Royalty Society of Medicine, 58,* 295-300.

Hoare, C. (1985). *Communicating Sequential Processes*. New York: Prentice-Hall.

Hoepfl, M. C. (1997). Choosing Qualitative Research: A Primer for Technology Education Researchers. *Journal of Technology Education, 9*(1), 1-13.

Available at: scholar.lib.vt.edu/ejournals/JTE/jte-v9n1/hoepfl.html

Horvitz, E. (1999). Principles of Mixed Initiative User Interfaces. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'99)* (pp. 159-166). New York: ACM.

Horvitz, E, Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. (1998). The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 256-265). San Francisco: Morgan Kaufmann.

Huber, G. P. (1983). Cognitive Style as a Basis for MIS and DSS Designs: Much Ado about Nothing? *Management Science, 29*(5), 367-379.

Huberman, A. M. & Miles, M. B. (1994). Data Management and Analysis Methods. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research* (pp. 428-444). London: Sage,.

Hume, D. (1888). *A Treatise of Human Nature* (A, Selby-Bigge, Ed.). Oxford, UK: Clarendon Press.

Hutchins, E. (1995). Cognition in the Wild. Cambridge, MA: MIT Press.

Introna, L. D. & Whitley, E. A. (1997). Against method-ism: Exploring the limits of method. *Information technology and people, 10*(1), 31-45.

Jacobson, B. (1992, April). The Ultimate User Interface. *Byte 17*(4), 175-182.

Jacobson, I., M. Christerson, et al. (1993). *Object-Oriented Software Engineering*: Wokingham, UK: Addison-Wesley.

Jagodzinski, A. P. & Holmes, S. H. (1989). Expert systems acceptability: human and organizational contexts of expert systems. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 226-241). Chichester, UK: Ellis Horwood.

James, D. T. D. (1990). Potential users' needs for information about expert systems. *Applied Ergonomics, 21*(3), 227-230.

Jarvenpaa, S. L., Ives, B. & Davis, G. B. (1991). Supply/demand of IS doctorates in the 1990s. *Communications of the ACM, 34*(1), 86-98.

Jeffery, D.R.; Votta, L.G. (1999). Empirical Software Engineering. *IEEE Transactions on Software Engineering, 25*(4), 435 –437.

Jeffries, R., Turner, A.T., Polson, P.G., & Atwood, M.E. (1981). The Processes Involved in Designing Software. In J.R. Anderson (Ed.) *Cognitive Skills and Their Acquisition*, Hillsdale, NJ: Lawrence Erlbaum, pp.255-283.

Johnson, H. & Johnson, P. (1993). Explanation Facilities and Interactive Systems. *Proceedings of the 1993 International Workshop on Intelligent User Interfaces* (pp. 159-166). New York: ACM.

Johnson, W. Lewis, (1991). Review of *Intelligent Tutoring Systems: Lessons Learned* by J. Psotka, L. D. Massey, & S. A. Mutter, (Eds.). *Artificial Intelligence, 48*(1), 125-134.

Johnson-Laird, P. N. (1983). *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Cambridge, MA: Harvard University Press.

Kaplan, A. (1964). *The conduct of inquiry*. Scranton, PA: Chandler.

Karsenty, L. (1996). An Empirical Evaluation of Design Rationale Documents. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)* (pp. 150-156). New York: ACM.

Kay, A. (1977, September). Microelectronics and the Personal Computer. *Scientific American, 237*(3), 230-244.

Keil, F. C. & Wilson, R. A. (2000). *Explanation and Cognition*. Cambridge, MA: MIT Press.

Keil, M., Beranek, P.M., & Konsynski, B.R. (1995). Usefulness and ease of use: field study evidence regarding task considerations. *Decision Support Systems*, 13, 75-91.

Kelle, U. (Ed.). (1995). *Computer-Aided Qualitative Data Analysis: Theories, Methods and Practice*. London: Sage.

Kemerer, C. F. & Slaughter, S. (1999). An Empirical Approach to Studying Software Evolution. *IEEE Transaction on Software Engineering, 25*(4), 493-509.

Kent, W. (1978). *Data and reality: Basic assumptions in data processing reconsidered*, Amsterdam: North-Holland.

Kerr, J. & Hunter, R. (1994). *Inside RAD*. Boston: McGraw Hill.

Kincaid, H. (1990). Assessing Functional Explanation in the Social Sciences. In A. Fine, M. Forbes, & L. Wessels (Eds.), *PSA 1990, 1*, 341-354. Reprinted in M. Martin & L. C. McIntyre (1994). (Eds.), *Readings in the Philosophy of Social Science* (pp. 415-428). Cambridge, MA: MIT Press.

Klein, H. K. & Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly, 23*(1), 67-94.

Klein, M. & Methlic, L. B. (1990). *Expert Systems: A Decision Support Approach.* Wokingham, UK: Addison-Wesley.

Klein, M. (1993). Capturing Design Rationale in Concurrent Engineering Teams. *IEEE Computer, 26*(9), 39-47.

Kling, R. (1991). Cooperation, coordination and control in computer-supported work. *Communications of the ACM, 34*(12), 83-88.

Kotterman, W. W. & Kumar, K. (1989). User Cube: A Taxonomy of End Users. *Communications of the ACM, 32*(11), 1313-1320.

Kraut, R. E. & Streeter, L. A. (1995). Coordination in Software Development. *Communications of the ACM, 38*(3), 69-81.

Kroes, P. (1998). Technological Explanations: The Relation Between Structure and Function of Technological Objects. *Society for Philosophy & Technology, 3*(3).

Available at: scholar.lib.vt.edu/ejournals/SPT/v3n3/html/KROES.html

KSL. (2000). The How Things Work Project, Stanford University, Available at: www-ksl.Stanford.edu/htw/htw-overview.html.

Kuwana, E. & Herbsleb, J.D. (1993). Representing knowledge in requirements engineering: an empirical study of what software engineers need to know. In *Proceedings of IEEE International Symposium on Requirements Engineering* (pp.273 –276).

Lamberti, D. M. & Wallace, W. A. (1990). Intelligent Interface Design: An Empirical Assessment of Knowledge Presentation in Expert Systems. *MIS Quarterly 14*(3), September, 279-311.

Landauer, T. K. (1987). Relations between Cognitive Psychology and Computer Systems Design. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 1-25). Cambridge, MA: MIT Press.

Langlotz, C. P. & Shortliffe, E. H. (1989). The critiquing approach to automated advice and explanation: rationale and examples. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 179-200). Chichester, UK: Ellis Horwood.

Latour, B. & Woolgar, S. (1986). *Laboratory life: The construction of scientific facts* (reprint ed.). Princeton, NJ: Princeton University Press.

Laurel, B. (1993). *Computers as Theatre*. Reading, MA: Addison-Wesley.

Lave, J. & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge, UK: Cambridge University Press.

Lee, A. S. (1989). A Scientific Methodology for MIS Case Studies. *MIS Quarterly, 13*(1), 33-50.

Lee, J. (1990). SIBYL: A Tool for Managing Group Design Rationale. In *Computer Supported Cooperative Work* (pp. 79-92). New York: ACM.

Lee, J. & Lai, K-Y. (1996). What's in Design Rationale? In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 21-51). Mahwah, NJ: Lawrence Erlbaum.

Lee, J. (1997). Design Rationale Systems: Understanding the Issues. *IEEE Expert: Intelligent Systems and Their Applications 12*(3), 78-85.

Lerch, F. J., Prietula, M. J., & Kulik, C. T. (1997). The Turing Effect: The Nature of Trust in Expert System Advice. In P. J. Feltovich, K. M. Ford, & R. R. Hoffman (Eds.), *Expertise in Context* (pp. 417-448). Menlo Park, CA: American Association for Artificial Intelligence.

Leung, K. S. & Wong, M. H. (1990). An expert system shell using structured knowledge: An object-oriented approach. *IEEE Computer, 23*(3), 38-47.

Lewis, C. (1988). Learning About Computers and Learning About Mathematics. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. Cambridge, MA: MIT Press.

Lewis, C., Rieman, J., & Bell, B. (1996). Problem-Centered Design for Expressiveness and Facility in a Graphical Programming System. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 1-19). Mahwah, NJ: Lawrence Erlbaum.

Liebowitz, J. (1997). Worldwide Perspectives and Trends in Expert Systems. *AI Magazine, 18*(2), 115-119.

Lim, K. H., Ward, L. M., & Benbasat, I. (1997). An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Tasks. *Information Systems Research, 8*(3), 254-319.

Lincoln, Y. S. & Guba, E. G. (1985). *Naturalistic Inquiry*. London: Sage.

Lindstone, H. & Turroff, H. (1975). *The Delphi Method: Technology and Applications*. Reading, MA: Addison-Wesley.

Lin, A. & Cornford, A. (2000). Sociotechnical perspectives on emergence phenomena. In E. Coakes, D. Willis, & R. Lloyd-Jones, (Eds.), *The New Sociotech: Graffiti on the Long Wall* (pp. 51-60). London: Springer Verlag.

Lipton, P. (1990). Contrastive Explanation. In D. Knowles (Ed.), *Explanation and Its Limits* (pp. 247-266). Cambridge, UK: Cambridge University Press.

Little, D. (1991). *Varieties of Social Explanation: An Introduction to the Philosophy of Social Science.* Boulder, CO: Westview Press.

Loeb, S. (1992). Architecting Personalised Delivery of Multimedia Information. *Communications of the ACM, 35*(12), 39-48.

Lofland, J. (1971). *Analyzing Social Settings: A Guide to Qualitative Observation and* Analysis. Belmont, CA: Wadsworth.

MacLean, A., Bellotti, V., & Young, R. (1990). What Rationale is there in Design? In D. Diaper, D. Gilmore, G., Cockton, & B. Shackel, B. (Eds.), *Proceedings of INTERACT '90: 3rd Conference on Human-Computer Interaction* (pp. 207-212). Amsterdam: Elsevier North Holland.

MacLean, A. & McKerlie, D. (1995). Design Space Analysis and Use-Representations, in *Scenario-Based Design: Envisioning Work and Technology in System Development.* J. M. Carroll (Ed.), New York: John Wiley.

MacLean, A., Young, R., & Moran, T. P. (1989). Design Rationale: The Argument Behind the Artifact? *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'89)* (pp. 247-252). New York: ACM.

MacLean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. (1996). Questions, Options, and Criteria: Elements of Design Space Analysis. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 21-51). Mahwah, NJ: Lawrence Erlbaum.

MacLean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. (1991). Design Space Analysis: Bridging from Theory to Practice with Design Rationale. Available as technical report EPC-1991-128, Rank Xerox Research Centre, Cambridge Laboratory..

Mahoney, C. (1997). Common Qualitative Methods. In J. Frechtling & L. Sharp (Eds.), *User-Friendly Handbook for Mixed Method Evaluations, NSF97-153.* National Science Foundation.
Available at: www.her.nsf.gov/HER/REC/pubs/NSF97-153.

Maida, A. S & Deng, M. (1989). A language to allow expert systems to have beliefs about their users In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 127-143). Chichester, UK: Ellis Horwood.

Mak, B. & Lyytinen, K. (1997). A Model to Assess the Behavioral Impacts of Consultative Knowledge Based Systems. *Information Processing & Management, 33*(4), 539-550.

Manning, A. D. (1999). Storytelling as a tool of technical explanation-improvisation risks and benefits. *Proceedings of the Professional Communication Conference, Communication Jazz: Improvising the New International Communication Culture, IEEE International*, 345-351.

Manning, P. K. & Cullum-Swan, B. (1994). Narrative, content, and semiotic analysis. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research* (pp.463-477). Thousands Oaks, CA: Sage.

March, S. T. & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems, 15*, 251-266.

Marcus, A. & van Dam, A. (1991). User Interface Developments for the Nineties. *Computer, 24*(9), 49-57.

Marcus, M. L. (1983). Power, Politics, and MIS Implementation. *Communications of the ACM, 26*(6), 430-444.

Marshall, C. & Rossman, G. B (1995). *Designing Qualitative Research* (2nd ed.). London: Sage.

Marshall, C. & Rossman, G. B (1999). *Designing Qualitative Research* (3nd ed.). Thousand Oaks, CA: Sage.

Mayer, R. E. (1988). Cognitive Aspects of Learning and Using a Programming Language. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 61-79). Cambridge, MA: MIT Press.

McCall, R., Mistrik, I., & Schuler, W. (1981) An integrated information and communication system for problem solving. *Proceedings of the Seventh International CODATA Conference*, (pp.107-115). London: Pergamon.

McCarthy, J. (1995). What has AI in Common with Philosophy? *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 2041-2044). San Francisco: Morgan Kaufmann.

Available at: www-formal.stanford.edu/jmc/

McGrath, J. E. (1995). Methodology Matters: Doing Research in the Behavioral and Social Sciences. In R. M. Baecker, J. Grudin, W. A. S. Buxton, & S. Greenberg (Eds.) *Readings in Human-Computer Interaction: Toward the Year 2000, 2nd Edition,* (pp.152-169) San Francisco: Morgan Kaugmann.

McKerlie, D., Preece, J., Jacques, R., Nonnecke, B., & MacLean, A. (1993). *Exploring the Design Space of Educational Hyper-Systems* (Tech. Rep. No. EPC-1993-112). Rank Xerox Research Centre, Cambridge, UK.

McKerlie, D. & MacLean, A. (1993). QOC in Action: Using Design Rationale to Support Design. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'93)* (p. 519). New York: ACM.

Miles, M. B. & Huberman, A. M. (1994). *Qualitative data analysis: an expanded sourcebook* (2nd ed.). London: Sage.

Miles, M. B. & Weitzman, E. A. (1994). Choosing Computer Programs for Qualitative Data Analysis. In M. B. Miles & A. M. Huberman (Eds.), *Qualitative Data Analysis: An Expanded Sourcebook* (2nd ed.). (pp. 311-317). London: Sage.

Miller, P. L. (1986). *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer.* New York: Springer-Verlag.

Miller, C. A. & Larson, R. (1992). An Explanatory and "Argumentative" Interface for a Model-Based Diagnostic System. *UIST: Proceedings of the ACM Symposium on User Interface Software and Technology* (pp. 43-52). New York: ACM.

Mills, H. D., Linger, R. C., & Hevner, A. R. (1987). Cleanroom Software Engineering. *IEEE Software, 4*(5), 19-25.

Minock, M. J. & chu, W. W. (1996). Explanation for Cooperative Information Systems. *Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems* (pp. 264-273). New York: Springer-Verlag.

Minsky, M. (1961). Steps toward artificial intelligence. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought* (pp. 406-450). New York: McGraw-Hill.

Mirel, B. (1998). Minimalism for Complex Tasks. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 179-218). Cambridge, MA: MIT Press.

Molich, R. & Nielsen, J. (1990). Improving a Human-Computer Dialogue. *Communications of the ACM, 33*(3), 338-348.

Moore, J. D. (1995). *Participating in Explanatory Dialogues: Interpreting and Responding to Questions In Context.* Cambridge, MA: MIT Press.

Moran, J. M. & Carroll, T. P. (1996). Overview of Design Rationale. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 1-19). Mahwah, NJ: Lawrence Erlbaum.

Moran, J. M. & Carroll, T. P. (Eds.). (1996). *Design Rationale: Concepts, Techniques and Use*. Mahwah, NJ: Lawrence Erlbaum.

Mostow, J. (1985). Towards Better Models of the Design Process. *The AI Magazine, 6*(1), 44-57.

Mumford, E., (1996). *Systems Design: Ethical Tools for Ethical Change*. Basingstoke, UK: Macmillan.

Murphy, G. C., Walker, R. J., & Baniassad, E. L. A. (1999). Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming. *IEEE Transactions on Software Engineering, 25*(4), 438-455.

Myers, M. D. (1997). Qualitative Research in Information Systems. *MIS Quarterly, 21*(2), 241-242.

MISQ Discovery, archival version, June, 1997,
Available at: www.misq.org/misqd961/isworld/

MISQ Discovery, updated version, April 28, 1999,
Available at: www.auckland.ac.nz/msis/isworld/.>

Nah, F. H. & Benbasat, I. (1997). Using Expert Support and its Explanation Facilities for Group Decision Making. *Proceedings of the Association for Information Systems 1997 Americas Conference* (pp. 988-990).

Nathan, M. J. (1990). Empowering the Student: Prospects for an Unintelligent Tutoring System. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'90)* (pp. 407-414). New York: ACM.

Naur, P. (1985). Programming as theory building. *Microprocessing and Microprogramming, 15*, 253-261.

Neches, R., Langley, P. & Klahr, D. (Eds.) (1987). *Production System Models of Learning and Development*. Cambridge, MA: MIT Press.

Newman, M. & Sabherwal, R. (1996). Determinants of Commitment to Information Systems Development: A Longitudinal Investigation. *MIS Quarterly, 20*(2), 23-54.

Newman, W. M. & Lamming, M G. (1995). *Interactive System Design*. Reading, MA: Addison-Wesley.

Norman, D. A. (1986). Cognitive Engineering. In D. A. Norman & S. W. Draper (Eds.), *User Centred System Design: New Perspectives on Human-Computer Interaction* (pp. 31-61). Hillsdale, NJ: Lawrence Erlbaum.

Norman, D. A. (1988). *The Psychology of Everyday Things*. New York: Basic Books.

O'Leary, D. E. (1998). Knowledge Management Systems: Converting and Connecting. *IEEE Intelligent Systems, 13*(3), 30-33.

Olson, G. M., Olson, J. S., Storroston, M., Carter, M., Herbsleb, J., & Rueter, H. (1996). The Structure of Activity During Design Meetings. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 217-240). Mahwah, NJ: Lawrence Erlbaum.

Oppenheim, A. N. (1992). *Questionnaire Design, Interviewing and Attitude Measurement* (2nd ed.). London: Pinter Publishers.

Orlikowski, W. J. (1988). *Information Technology and Post-Industrial Organizations: An Examination of the Computer-Mediation of Production Work.* Unpublished Ph.D. Thesis, Stern School of Business, New York University.

Orlikowski, W. (1993). CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development. *MIS Quarterly, 17*(3), 309-340.

Orlikowski, W. (1996). Evolving with notes: Organizational change around groupware technology. In C. U. Ciborra (Ed.), *Groupware & teamwork: Invisible aid or technical hindrance,* Chichester: Wiley, 23-59.

Pandit, N. R. (1996). The Creation of Theory: A Recent Application of the Grounded Theory Method. *The Qualitative Report, 2*(4). www.nova.edu/ssss/QR/QR2-4/pandit.html

Parnas, D. L., & Clements, P. C. (1986). A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering, 12*(2), 251-257.

Passini, R. (1984). *Wayfinding in Architecture.* New York: Van Norstrand Rheinhold.

Patrick, A. & McGurgan, A. (1993). One Proven Methodology for Designing Robust Online Help Systems. *Proceedings of the 11th annual international conference on systems documentation* (pp. 223-232). New York: ACM.

Patton, M. Q. (1990). *Qualitative Evaluation and Research Methods* (2nd ed.). London: Sage.

Paul, R. J. (1994). Why Users Cannot 'Get What They Want'. *International Journal of Manufacturing Systems Design, 1*(4), 389-394.

Paxton, A. L. & Turner, E. J. (1984). The application of human factors to the needs of the novice computer user. *Int. J. Man-Machine Studies, 20*, 137-156.

Pearl, J. (1996). *The Art and Science of Cause and Effect.* Lecture given as part of the UCLA 81st Faculty Research Lecture Series. UCLA Computer Science Department Tech. Rep. R-248.

Petroski, H. (1996). *Invention by Design: How Engineers Get from Thought to Thing.* Cambridge, MA: Harvard University Press.

Pfleeger, S. L. (1999). Albert Einstein and Empirical Software Engineering. *IEEE Computer*, October, 32-38.

Pitt, J. (1988). *Theories of Explanation.* Oxford, UK: Oxford University Press.

Pitt, J. C. (2000). *Thinking About Technology: Foundations from the Philosophy of Technology.* New York: Seven Bridges Press.

Pollard, P. & Crozier, R. (1989). The Validity of Verbal Reports: unconscious processes and biases in judgement. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 13-57). Chichester, UK: Ellis Horwood.

Polson, P. G. & Lewis, C. H. (1990). Theory-based design for easily learned interfaces, *Human Computer Interaction, 5*, 191-220.

Poore, J. H. & Mills, H. D. (1989). An overview of the Cleanroom software development process. *Proceedings of the ACM International Workshop on Formal Methods in Software Development.* New York: ACM.

Potts, C. (1996). Supporting Software Design Methods and Design Rationale. In J. M. Moran & T. P. Carroll (Eds.), *Design Rationale: Concepts, Techniques and Use* (pp. 295-321). Mahwah, NJ: Lawrence Erlbaum.

Potts, C., Takahashi, K., & Anton, A. (1994). Inquiry-based requirements analysis. *IEEE Software, 11*(2), 21-32.

Pouloudi, A. & Whitley, E. A. (1997). Stakeholder identification in inter-organizational systems: Gaining insights for drug use management systems. *European Journal of Information Systems, 6*(1), 1-14.

Pratt, T. W. & Zelkowitz, M. V. (2000). *Programming Languages: Design and Implementation.* Upper Saddle River, NJ: Prentice-Hall.

Prein, G. & Kuckartz, U. (1995). Computers and Triangulation: Introduction. In U. Kelle, (Ed.),*Computer-Aided Qualitative Data Analysis: Theories, Methods and Practice* (pp. 152-157). London: Sage.

Pressman, R.S. (2001). *Software Engineering: A Practitioner's Approach.* Boston: McGraw Hill.

Putnam, H. (1978). *Meaning and the Moral Sciences.* London: Routledge & Kegan Paul.

Railton, P. (1988). A Deductive-Nomological Model of Probabilistic Explanation. In J. C. Pitt (Ed.), *Theories of Explanation* (pp.119-135). Oxford, UK: Oxford University Press.

Ramesh, B. & Dhar, V. (1994). Representing and Maintaining Process Knowledge for Large-Scale Systems Development. *IEEE Expert, 9*(4), 54-59.

Ramesh, B. & Jarke, M. (2001). Towards Reference Models for Requirements Traceability, *IEEE Transactions on Software Engineering, 27*(1), 58-93.

Ramesh, B. & Sengupta, K. (1995). Multimedia in a design decision support system. *Decision Support Systems, 15*(3), 181-196.

Randall, N. & Pedersen, I. (1998). Who Exactly is Trying to Help Us? The Ethos of Help Systems in Popular Computer Applications. *Proceedings of the 16th annual international conference on computer documentation* (pp. 63-69). New York: ACM.

Rawson, P. (1987). *Creative Design: A New Look at Design Principles.* London: Macdonald Orbis.

Redmiles, D. F. (1993). Observations On Using Empirical Studies in Developing a Knowledge-Based Software Engineering Tool. *Proceedings of the Eighth Knowledge-Based Software Engineering Conference*, Page(s): 170 -177.

Remenyi, D. & Williams, B. (1996). The nature of research: qualitative or quantitative, narrative or paradigmatic? *Information Systems Journal, 6*, 131-146.

Rettig, M. (1991). Nobody Reads Documentation. *Communications of the ACM, 34*(7), 19-24.

Rettig, M. (1992) Hat Racks for Understanding. *Communications of the ACM, 35*(10), 21-24.

Rheingold, H. (2000). *Tools for Thought: The History and Future of Mind-Expanding Technology, MIT Press Edition.* Cambridge, MA: MIT Press.

Rich, E. & Knight, K. (1991). *Artificial Intelligence* (2nd ed.). New York: McGraw Hill.

Richards, T. & Richards, L. (1995). Using Hierarchical Categories in Qualitative Data Analysis. In U. Kelle (Ed.), *Computer-Aided Qualitative Data Analysis: Theories, Methods and Practice* (pp.80-95). London: Sage.

Rittel, H. (1984). Second Generation Design Methods. In N. Cross (Ed.), *Developments in Design Methodology* (pp. 317-327). New York: John Wiley.

Rittel, H. & Webber, M. (1973). Dilemmas in a General Theory of Planning. *Policy Science, 4*, 155-169.

Robb, F. and Brown, T. (1987, June). The machine intelligence family. *The Accountant's Magazine*, 50-53.

Robey, D. (1996). Research Commentary: Diversity in Information Systems Research: Threat, Promise, and Responsibility. *Information Systems Research*, 7(4), 400-408.

Robson, C. (1994). *Experiment, Design and Statistics in Psychology* (3rd ed.). London: Penguin.

Roesler , A. W. & McLellan, S. G. (1995). What help do users need?: Taxonomies for on-line information needs access methods. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)* (pp. 437-441). New York: ACM.

Rosenberg, A. (1995). *Philosophy of Social Science* (2nd ed.). Oxford, UK: Westview Press.

Rosson, M. B., Maass, S., & Kellogg, W. A. (1988). The Designer as User: Building Requirements for Design Tools from Design Practice. *Communications of the ACM, 31*(11), 1288-1298.

Roth, Steven F., Mattis, J. & Mesnard, X. (1991). Graphics and Natural Language Components of Automatic Explanation. In J. W. Sullican & S. W. Tyler (Eds.), *Intelligent User Interfaces*. Reading, MA: Addison-Wesley.

Royce, W.W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *Proceedings of IEEE WESCON*, August.

Ruben, D.-H. (1990). *Explaining Explanation*. London: Routledge.

Ruben, D.-H. (Ed.). (1993). *Explanation*. Oxford, UK: Oxford University Press.

Ruben, D.-H. (1998). The Philosophy of the Social Sciences. In *Philosophy 2: Further Through the Subject* (pp.420-469). Oxford, UK: Oxford University Press.

Salmon, W. (1975). Theoretical Explanation. In S. Korner (Ed.), *Explanation* (pp. 118-145). Oxford, UK: Oxford University Press.

Salmon, W. (1984). *Scientific Explanation and the Causal Structure of the World.* Princeton, NJ: Princeton University Press.

Salmon, W.C. (1993). Scientific Explanation and the Causal Structure of the World. In D.-H. Ruben (Ed.), *Explanation* (pp.78-112). Oxford, UK: Oxford University Press.

Salmon, W.C. (1998). *Causality and Explanation.* New York: Oxford University Press.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action.* New York: Basic Books.

Schuler, D., & Namioka, A., (Eds.). (1993), *Participatory Design: Principles and Practice.* Hillsdale, NJ: Lawrence Erlbaum.

Scriven, M. (1956). A Possible Distinction between Traditional Scientific Disciplines and The Study of Human Behavior. In *Minnesota Studies in the Philosophy of Science, v.1* (pp.330-339). University of Minnesota Press. Reprinted in M. Martin & L. C. McIntyre (Eds.). (1994), *Readings in the Philosophy of Social Science* (pp. 71-77). Cambridge, MA: MIT Press.

Scriven, M. (1988). Explanations, Predictions, and Laws. In J. Pitt (Ed.), *Theories of Explanation.* Oxford, UK: Oxford University Press.

Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering, 25*(4), 557-572.

Searle, J. (1969). *Speech Acts: An Essay in the Philosophy of Language.* Cambridge, UK: Cambridge University Press.

Searle, J. R. (1995). *The Construction of Social Reality.* New York: Basic Books.

Seely Brown, J. (1986). From Cognitive to Social Ergonomics and Beyond. In D. A. Norman & S. W. Draper (Eds., *User Centered System Design.* Hillsdale, NJ: Lawrence Erlbaum.

Seely Brown, J. & Duguid, P. (2000). *The Social Life of Information.* Boston, MA: Harvard University Press.

Sell, P.S. (1985). *Expert Systems - A Practical Introduction.* London: MacMillan.

Seidel, J. & Kelle, U. (1995). Different Functions of Coding in the Analysis of Textual Data. In U. Kelle (Ed.), *Computer-Aided Qualitative Data Analysis: Theories, Methods and Practice* (pp. 52-61). London: Sage.

Shank, R.C. (1986). *Explanation Patterns.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Shipman, F. & McCall, R. (1997). Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing (AIEDAM), 11*(2), 141-154.

Shneiderman, B. (1979). Human factors experiments in designing interactive systems. *IEEE Computer, 12*(12), 9-24.

Shneiderman, B., Byrd, D., & Croft, W. B. (1997). Sorting Out Searching: A User-Interface Framework for Text Searches. *Communications of the ACM, 41*(4), 95-98.

Shneiderman, B. (2000). The Limits of Speech Recognition. *Communications of the ACM, 43*(9), 63-65.

Shum, S. J. (1991). *A Cognitive Analysis of Design Rationale Representation.* Unpublished Ph.D. Thesis, Department of Psychology, University of York.

Shum, S., MacLean, A., Forder, J. and Hammond, N. V. (1993). Summarising the Evolution of Design Concepts Within a Design Rationale Framework. *Adjunct Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'93)*, 43-44. New York: ACM.

Siemer, J. & Angelides, M. C. (1994). Embedding An Intelligent Tutoring System In A Business Gaming-Simulation Environment. *Proceedings of the 1994 Winter Simulation Conference* (pp. 1399-1406). New York: ACM.

Silverman, B. G. (1992). Survey of expert critiquing systems: practical and theoretical frontiers. *Communications of the ACM, 35*(4), 106-127.

Silvestro, K. (1988). Using explanations for knowledge-base acquisition, *Int'l. J. Man-Machine Studies, 29*, 159-169.

Silverman, D. (1993). Interpreting Qualitative Data: Methods for Analysing Talk, Text, and Interaction. London: Sage.

Simon, H. A. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, MA: MIT Press.

Sjöberg, C & Timpka, T (1995). Inside multi-disciplinary design in medical informatics: experiences from the use of an argumentative design method. *Proceedings of MEDINFO'95 Tri-annual World Conference in Medical Informatics*. Amsterdam: Elsevier.

Slagle, J. R., Gardiner, D. A., & Han, K. (1990). Knowledge Specification of an Expert System. *IEEE Expert, 5*(4), 29-38.

Sleeman, D. & Brown, J. S. (1982). Introduction: Intelligent Tutoring Systems. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 1-11). London: Academic Press.

Smith, D (1998). Computerizing Computer Science. *Communications of the ACM 41*(9), 21-23.

Sperber, D. (1996). *Explaining Culture: A Naturalistic Approach.* Oxford, UK: Blackwell.

Spinuzzi, C. & Zachry, M. (2000). Genre Ecologies: An Open-System Approach to Understanding and Constructing Documentation. *ACM Journal of Computer Documentation, 24*(3), 169-181.

Spinuzzi, C. (1999). Grappling with Distributed Usability: A Cultural-Historical Examination of Documentation Genres Over Four Decades. *Proceedings of the 17th annual international conference on computer documentation* (pp. 16-21). New York: ACM.

Spivey, J. M. (1989). *The Z Notation: A Reference Manual.* Hemel Hempstead, UK: Prentice-Hall.

Stake, R. E. (1998). Case Studies. In N. K. Denzin & Y. S. Lincoln (Eds.), *Strategies of Qualitative Inquiry* (pp.86-109). Thousand Oaks, CA: Sage Publications.

Stefik, M. & Conway, L. (1982). Towards the Principled Engineering of Knowledge. *The AI Magazine, 3*(3), 4-16.

Stevens, C. (1992). Automating the creation of information filters. *Communications of the ACM, 35*(12), 48.

Stille, A. (1999, March 8). Overload. *The New Yorker*, 38-44.

Strasser, S. (1985). *Understanding and Explanation.* Pittsburgh, PA: Duquesne University Press.

Stratton, P. (1997). Attributional coding of interview data: Meeting the needs of long-haul passengers. In N. Hayes (ed.) *Doing Qualitative Analysis in Psychology* (pp.115-142). Hove, U.K.: Psychology Press.

Strauss, A. & Corbin, J. (1994). Grounded Theory Methodology: An Overview. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research* (pp. 273-285). London: Sage.

Strauss, A. & Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (2nd ed.). Thousand Oaks, CA: Sage.

Stylianou, A. C., Madey, G. R., & Smith, R. D. (1992). Selection Criteria for Expert System Shells: A Socio-Technical Framework. *Communications of the ACM, 35*(10), 30-48.

Suchman, L. (1987). *Plans and Situated Actions: The problem of human machine communication.* Cambridge, UK: Cambridge University Press.

Sudman, S. & Bradburn, N. M. (1982). *Asking Questions: A Practical Guide to Questionnaire Design.* London: Jossey-Bass.

Suthers, D. (1993). Preferences for Model Selection in Explanation. *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (pp. 1208-1213). San Francisco: Morgan Kaufmann.

Swartout, W. R. (1980). *Producing Improved Explanations and Justifications of Expert Consulting Programs Using an Automatic Program Generator.* Ph.D. Thesis, MIT, Cambridge, MA.

Swartout, W. R. (1983). XPLAIN: A System for Creating and Explaining Expert Consulting Programs. *Artificial Intelligence, 21*(3), 285-325.

Swartout, W. R. & Smoliar, S. W. (1987). On making expert systems more like experts. *Expert Systems, 4*(3), 196-207.

Swartout, W. R., Paris, C. & Moore, J. D. (1991). Design for Explainable Expert Systems. *IEEE Expert, 6*(3), 58-64.

Swartout, W. R. (1996). Future Directions in Knowledge Based Systems. *ACM Computing Surveys, 28*(4). New York: ACM.

Tanik, M. M. & Yeh, R. T. (1989). Guest Editors' Introduction: Rapid Prototyping in Software Development. *IEEE Computer, 22*(5), 9-10.

Tanner, M. C. & Keuneke, A. M. (1991). The Roles of the Task Structure and Domain Functional Models. *IEEE Expert, 6*(3), 50-57.

Tazi, S. & Novick, D. G. (1998). Design Rationale for Complex System Documentation. *Proceedings of the Conference on Complex Systems, Intelligent Systems and Interfaces.* Lettres de l'Intelligence Artificielle, combined volumes 134-236, 49-51.

Teach, R. L. & Shortliffe, E. H. (1981). An analysis of physician attitudes regarding computer-based clinical consultation systems. *Computers and Biomedical Research, 14*, 542-558.

Thagard, P. (1999). Explanation. In R. A. Wilson & F. Keil, (Eds.), *The MIT Encyclopedia of Cognitive Science*, (pp. 300-301). Cambridge, MA: MIT Press.

Available at: cognet.mit.edu/MITECS/

Todd, F. J. & Hammond, K. R. (1965). Differential Effects in Two Multiple-cue Probability Learning Tasks. *Behavioral Science, 10*, 429-435.

Torkzadeh, R., Pflughoeft, K., & Hall, L. (1999). Computer self-efficacy, training effectiveness and user attitudes: an empirical study. *Behaviour & Information Technology, 18*(4), 299-309.

Toulmin, S. (1958). *The Uses of Argument.* Cambridge, UK: Cambridge University Press.

Trauth, E. M. & Jessup, L. M. (2000). Understanding Computer-Mediated Discussions: Positivist and Interpretive Analyses of Group Support System Use. *MIS Quarterly, 24(1).* 43-79.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information.* Cheshire, CT: *Graphics* Press.

Tufte, E. R. (1990). *Envisioning Information.* Cheshire, CT: Graphics Press.

Tufte, E. R. (1997). *Visual explanations: images and quantities, evidence and narrative.* Cheshire, CT: Graphics Press.

Turk, K. L. & Nichols, M. C. (1996). Online Help Systems: Technological Evolution or Revolution? *Proceedings of the 14th annual international conference on Marshaling new technological forces: building a corporate, academic, and user-oriented triangle* (pp. 239-242). New York: ACM.

Turner, J. A. (1987). Understanding the Elements of System Design. In R. J. Boland & R. H. Hirschheim (Eds.), *Critical Issues in Information Research*, (pp. 97-111). Chichester, UK: John Wiley and Sons.

U.S. Department of Defense (1987). *Human engineering procedures guide* (DoD-HDBK-763). Washington D.C.: US Department of Defense.

Van der Meij, H. (1992). A Critical Assessment of the Minimalist Approach to Documentation. *Proceedings of the 10th annual international conference on systems documentation* (pp. 7-17). New York: ACM.

van Fraassen, B. C. (1988). The Pragmatic Theory of Explanation. In J. Pitt (Ed.), *Theories of Explanation* (pp. 135-155). Oxford, UK: Oxford University Press.

van Fraassen, B. (1991). The Pragmatics of Explanation. In R. Boyd, P. Gasper, & J. D. Trout (Eds.), *The Philosophy of Science* (pp. 317-327). Cambridge, MA: MIT Press.

Vasandani, V. & Govindaraj, T. (1995). Knowledge Organization in Intelligent Tutoring Systems for Diagnostic Problem Solving in Complex Domains. *IEEE Transactions on Systems, Man, and Cybernetics, 25*(7), 1076-1096.

Vessey, I. & Sravanapudi, P. (1995). CASE Tools as Collaborative Support Technologies. *Communications of the ACM, 38*(1), 83-95.

Vincenti, W. G. (1990). *What Engineers Know and How They Know It.* Baltimore: John Hopkins University Press.

Vliegen, H. J. W. & Van Mal, H. H. (1990). Rational Decision Making: Structuring of Design Meetings. *IEEE Transactions on Engineering Management, 37*(3), 185-190.

Walsham, G. (1995). The Emergence of Interpretivism in IS Research. *Information Systems Research, 6*(4), 376-394.

Wells, A. J. (1998). Turing's Analysis of Computation and Theories of Cognitive Architecture. *Cognitive Science, 22*(3), 269-294.

Weiner, J. L. (1989). The effect of user models on the production of explanations. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language* interface (pp. 144-156). Chichester, UK: Ellis Horwood.

Weitzman, E. & Miles, M. B. (1994). *Computer programs for qualitative data analysis.* London: Sage.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems.* Los Altos, CA: Morgan Kaufmann.

Whitley, E. A. (1990). Expert Systems: True support for the process of decision making. *Proceedings of the International Conference of the Special Interest Group on Business Data Processing (SIGBDP),* (pp. 123-140). New York: ACM.

Whitley, E. A. (1998). Method-ism in practice: Investigating the relationship between method and understanding in web page design. In R. Hirschheim, M. Newman, & J. I. D. Gross (Eds.), *Proceedings of the 19th International Conference on Information Systems.* Helsinki, Finland, ICIS, 68-75.

Wick, M. R. & Slagle, J. R. (1989). An Explanation Facility for Today's Expert Systems. *IEEE Expert 4*(1), 26-36.

Wick, M. R. & Thompson, W. B. (1992). Reconstructive expert system explanation. *Artificial Intelligence, 54*(1), 33-70.

Winograd, T. & Flores, F. (1986). *Understanding Computers and Cognition: A New Foundation for Design.* Norwood, NJ: Ablex Publishing Corporation.

Winograd, T. (1995). Forward. In W. M. Newman & M. G. Lamming (Eds.), *Interactive System Design* (pp. xx-xx). Reading, MA: Addison-Wesley.

Winston, P. H. (1984). Preface. In P. H. Winston (Ed.), *The AI Business: The Commercial Uses of Artificial Intelligence.* Cambridge, MA: MIT Press.

Wixon, D. (1995). Qualitative Research Methods in Design and Development. *Interactions, 11*(4), 19-24.

Wixon, D & Ramey, J. (1996) *Field Methods Casebook for Software Design.* New York: John Wiley.

Wong, B. K. & Monaco, J. A. (1995). A bibliography of expert system applications for business (1984-1992). *European Journal of Operational Research, 85*, 416-432.

Woolf, B. P. (1996). Intelligent Multimedia Tutoring Systems. *Communications of the ACM, 39*(4), 30-31.

Wu, A. K-W. (1993). Paradigms for ITS (Intelligent Tutoring System), Computer, *Proceedings of TENCON '93: Communication, Control and Power Engineering* (pp. 96 -99). IEEE.

Ye, L. R. & Johnson, P. E. (1995). The Impact of Explanation Facilities on User Acceptance of Expert Systems Advice. *MIS Quarterly, 19*(2), 157-172.

Yin, R. K. (1984). *Case Study Research: Design and Methods, Revised Edition,* Newbury Park, CA: Sage.

Yoon, Y., Guimaraes, T., & O'Neal, Q. (1995). Exploring the Factors Associated With Expert Systems Success. *MIS Quarterly, 19*(1), 83-106.

Yourdon E. (1989). *Modern structured analysis.* Upper Saddle River, NJ: Prentice-Hall.

Zachry, M. (1999). Constructing usable documentation: a study of communicative practices and the early uses of mainframe computing in industry. *Proceedings of the 17th annual international conference on Computer documentation* (pp. 22-25). New York: ACM.

Zimmer, A. C. (1989). The conceptualization of explanatory intervention in a dynamic human-computer interaction. In C. Ellis (Ed.), *Expert Knowledge and Explanation: the knowledge-language interface* (pp. 157-178). Chichester, UK: Ellis Horwood.

Appendix A – Study 1 Interview Guides

Interview Guide Version 1

A. Questions related to generation and use of design documentation

1. What kinds of system projects do you work on and what is your role?
2. Do you work with directly with users?
3. How do you capture and document system requirements and design information for your projects e.g., meeting agendas and minutes, e-mail, formal specifications?
4. Do you use a design formalism, set of document templates, etc. to capture requirements and design information?
5. How is design information later used?
6. Do you classify and prioritise system requirements and design information?
7. Do you have problems with the generation, capture and use of requirements and design information and if so, what kinds of problems?

B. Questions related to user access to system information

1. How do you provide users with system information e.g., on-line help, manuals, etc?
2. What kind of information do you think constitutes an *explanation* of a system feature or process?
3. What are the most important elements of a system feature or process explanation?
4. Do you feel users have access to adequate information about your projects?
5. Do different types of users require different kinds of system information?
6. What information, not currently provided to users, might help them with their use of systems?
7. Are there types of system design information that you feel should NOT be provided to users?

C. Questions related to providing system analysis and design information to users

1. Do users have access to system *analysis* and *design* information?
2. If not, do you feel that users would benefit from access to system analysis and design information?
3. Do you think information from the analysis and design phase of a project can contribute to explanations of system features
4. Do you think users are interested in the different design alternatives that are considered, and why certain alternatives are selected over others?
5. How can system design information be transformed to make it more useful to users?

Interview Guide Version 2

A. General

1. How long have you been in the industry?
2. What kinds of roles have you had?
3. What kinds of system projects do you work on?
4. Do you work with directly with users?

B. Questions related to generation and use of design documentation

1. How do you capture and document requirements & design information for your projects e.g., meeting agendas and minutes, e-mail, formal methodology, documentation templates?
2. How is analysis and design information used later in the project and after implementation?
3. Do you have problems with the generation, capture and use of requirements & design information and if so, what kinds of problems?
4. How much of the information generated during analysis & design relates to the actual use of the system?
5. Do you think there are benefits to capturing more, or less, information during analysis & design?

C. Questions related to user access to system information

1. How do you provide users with system information e.g., on-line help, manuals, etc?
2. What kind of information do you think constitutes an *explanation* of a system feature or process?
3. What are the most important elements of a system feature or process explanation?
4. Do you feel users have access to adequate information about your projects?
5. Do different types of users require different kinds of system information?
6. What information, not currently provided to users, might help them with their use of systems?
7. Are there types of system design information that you feel should NOT be provided to users?

D. Questions related to providing system analysis and design information to users

1. Do users have access to information produced during the system *analysis* and *design* phases of your projects? If so, how do they use it?
2. If not, do you feel that users would benefit from access to system analysis and design information?
3. Do you think information from the analysis and design phases of a project can contribute to explanations of system features?
4. Do you think users are interested in the different design alternatives that are considered, and why certain alternatives are selected over others?
5. How can system analysis & design information be transformed to make it more useful to users?

E. Explain the thesis and probe reaction.

Appendix B – Study 1 Interview Participants

Table 10 - Study 1 Interview Participants

Name	Organisation	Role	Date
David Griffin	Cognos Guildford, Surrey, U.K.	Director, Product Development	20 April 1999
Nick Rich	Andromedia Europe Ltd.	European Project Manager	30 April 1999
Stephen Hester	Apple Computer Inc. Uxbridge, Mddx., U.K.	Director, European Information Systems & Technology	4 May 1999
Raymond Cansick	Ernst & Young London, U.K.	Global Manager, Notes/Domino Operations	7 May 1999
David Colewell	Ernst & Young London, U.K.	Technology Business Manager	7 May 1999
Simon Shearston	Microsoft Corporation Reading, UK	Enterprise Program Manager	13 May 1999
Clive Philpott	Concert Management Services Inc. (BT subsidiary) San Jose, CA, USA	Application Development Manager	13 July 1999
Barry O'Kane	Concert Management Services Inc. (BT subsidiary) San Jose, CA, USA	Contract Software Developer	13 July 1999
Tom Proulx	Netpulse Communications San Francisco, CA, USA	Chief Executive Officer	15 July 1999
Jerry Peeks	Scriptics Corporation Mountain View, CA, USA	Documentation/Training Developer	14 July 1999
Helen Shaw	Apple Computer Inc. Cupertino, CA, USA	Support & Development Manager, Financial Systems	14 July 1999
Lori Gordon	Oddessa Solutions San Jose, CA, USA	Enterprise Data Modeler	14 July 1999
Tim Danison	Forte Software Oakland, CA, USA	Development Manager	15 July 1999
Ken Considine	Concert Management Services Inc. (BT subsidiary) San Jose, CA, USA	Senior Developer	15 July 1999

Name	Organisation	Role	Date
Irene Kung	Adobe Systems, Inc. San Jose, CA, USA	Application Development Manager	16 July 1999
Harold Brown	Knowmads Consulting, Inc. Littleton, MA, USA	Principal Consultant	20 July 1999
Mike Vivino	CCOM Information Systems Iselin, NJ, USA	Project Manager	20 July 1999
Eileen Kramer	Direct Report, Inc. Maynard, MA, USA	Web Developer	21 July 1999
Ron Gruner	Direct Report, Inc. Maynard, MA, USA	Chief Executive Officer	21 July 1999
Mike Rowe	Harvard University University Information Systems Cambrdge, MA. USA	Systems Analyst	22 July 1999
John Walters	Sapient Corporation Cambridge, MA, USA	Project Manager	22 July 1999
Ed Leung	I2 Technologies, Inc. Cambridge, MA, USA	Manager, Optimisation	22 July 1999
Stephan Kolitz	Draper Laboratory Cambridge, MA, USA	Principal Member Technical Staff	22 July 1999
Peter McLeod	General Electric Corp., Harris Energy Control Systems, UK, Ltd. Livingston, Scotland, UK	Director of Product Development	25 Sept 1999
Hector MacLean	Oracle Corporation Reading, UK	Human Resources/Payroll Development	9 Oct 1999
David Levy	General Magic, Inc. Mountain View, CA, USA	Manager, Operating Systems Platform Marketing (previous position, currently Ph.D. student, Kings College London)	15 Feb 2000
David Nicholas	Stafford Trading, Ltd. London, UK	Senior Software Engineer	6 July 2000

Appendix C – Study 1 Conceptual Framework

Pre-empirical Conceptual Framework

Following Miles and Huberman (1994), the initial, pre-empirical set of codes was developed based on the conceptual framework embodied in the research questions and in the interview questionnaire (version 1, see Appendix B), which were derived from the literature. This initial code set appears in the table below. Each code includes references to the research questions (Q1, Q2, etc.) and interview guide questions (A1, B3, etc.) to which they relate. Once again, the research questions are:

Q1. What can philosophical theories of explanation contribute to the development of a framework for integrated explanation facilities?

Q2. Can ideas from the fields of explanation systems and design rationale be integrated into the framework from Q1 to produce an implementable model for explanations of information systems?

Q3. Is the model from Q2 operationally realistic, is it cost-effective, and can it be integrated into the IS development process?

Table 11 - Pre-empirical Code Set

Explanation	These codes relate to views expressed on the nature of explanation and the pragmatic issues that arise in attempting to provide explanations.
EXP-STRUCTURE Q1, Q2 B2, B3, C3, C5	A structural view of explanation. What are the components and the relationships between components in an explanation.
EXP-FUNCTION Q1, Q2, B2, B3, C3, C5	The function or purpose of explanation.
EXP-CONTENT Q1, Q2, B2, B3, B6, C2, C3, C4, C5	Issues related to explanation content.
EXP-NORMATIVE Q1 B2, B3, C3, C5	Ideal explanation forms.
EXP-EXAMPLE Q1, Q2	Examples of explanations related to IS.

B2, B3, C3	
EXP-PROBLEM Q1, Q2, C1, C2, C3	Problems providing explanations in IS.
Design Rationale	These codes relate to views expressed on capture and use of design rationale information in the IS lifecycle.
DRL-EXPLAIN Q2 B2, B3, C1, C2, C3, C4	Issues on the relationship between DR and explanation.
DRL-CAPTURE Q2, Q3 A3, A4, A6, A7	Issues that arise in capturing DR.
DRL-USE Q2, A5, B4, C1, C3, C4	Issues related to how DR information is used.
DRL-ROI Q2, Q3 A5, A7, B3, B4, C2, C3, C4	The perceived value and costs (Return on Investment) of DR information.
Development Practices	These codes relate to aspects of the IS lifecycle.
DEV-CAPTURE Q2, Q3 A3, A6	Issues related to capture of analysis and design information in the project lifecycle.
DEV-DOCUSE Q2 A5, B1, B4, B6, B7, C1, C2, C3	Issues related to how analysis and design information is used in the IS lifecycle.
DEV-TEAM Q2 A3, A3, A5, A7, B4, B7, C1	Issues related to the dynamics of the development team.
DEV-ORG Q2, Q3 A4, A5, B7	Issues related to how the organisational context impacts the IS lifecycle.
DEV-USERS Q2, Q3 A2, A5, A7, B1, B4, B5, B7, C2, C4	Issues related to the role of users or customers in the IS lifecycle.
DEV-PROBLEMS Q2, Q3 A7, B4, B6, B7	Issues related to the problems that arise in using information generated during analysis and design.
Usability	These codes relate to views expressed on factors that enhance or detract from IS usability.

USB-EXPLAIN Q1, Q2 B1, B4, B6, C1, C2	Issues related to the role of explanations in the usability of IS.
USB-DR Q2, Q3 B1, B4, B6, C1, C2, C3	Issues related to the role of DR information in the usability of IS.
USB-USERTYPES Q2 B5	Issues related to different classes of users and how these differences impact information needs.
USB-DOC Q2 B1	Issues related to the role of documentation in IS usability.
USB-HELP Q2 B1	Issues related to the role of online help in IS usability.
USB-TRAIN Q2 B1	Issues related to the role of training in IS usability.
Utility	These utility codes have miscellaneous roles including classifying information on participant project roles, project types, experience level, etc.
UTL-ROLE A1	Used to designate the role of the interview participant. To be sub-typed.
UTL-PROJTYPE A1	Used to designate the type of projects the interview participant in involved with. To be sub-typed.
UTL-EXPERIENCE A1	Used to designate the level of experience of the interview participant. To be sub-typed.
UTL-USER A2	Used to designate the interview participant's level of user contact. To be sub-typed.

Evolution of the Conceptual Framework

The first six interviews were first coded using the baseline code set. Where necessary, new codes were added or existing codes were sub-typed. The next step was to review the densities of each of the codes. Codes with low densities were analysed with the goal of either eliminating the code or merging it with another. This process involved

reviewing the transcripts and focusing only on the codes with low densities. Next, the quotations for each of the codes with high densities were analysed to determine whether these codes should be split into more fine-grained units. This stage of analysis corresponded to roughly to the open coding phase prescribed by the grounded theory approach.

One difficulty encountered at this stage of the analysis was distinguishing between what Seidel and Kelle (1995) describe as indexing versus summarising codes. Indexing codes act as sign-posts in that they simply denote that a certain text passage assigned with the code includes a discussion related to the concept represented by the code. Summarising codes, on the other hand, are used to represent a fact about the text passage. For example, a code such as DRL-ROI-POS might denote that a given text passage includes a statement in support of the design rationale approach based on its return on investment. Richards and Richards (1995) refer to these two types of code functions as referential versus factual. Both types of code functions were applied though in the earlier stages of analysis, indexing or referential codes were more useful based on their more microscopic view of the data.

The second phase of analysis included the development of a set of categories into which various codes, or concepts, were organised. These higher level categories were further divided according to three general principles: core theoretical concepts, factors, and 'utility' codes. Core theoretical concepts were derived from the theoretical framework and were focussed on aspects of explanation, design rationale, and the relationships between them. Factors were derived from the text of the interviews and are considered to influence the conceptual notions in some significant way. Examples of factors include issues such as cost, use of development formalisms, and training, among many others. Utility codes were used to identify the role of interview participants, organisation types, the industry, and others. The distinction between utility codes and factors was largely a matter of convenience, a utility code was given the same consideration in terms of their potential significance in the overall framework. For example, number of years IS development experience is a utility code, but could act as a factor in certain circumstances.

For example, consider a response to one of the questions related to capturing design rationale. A given text fragment from the response might be coded as DRL-

CAPTURE (relating to capture of design rationale), FAC-ACCOUNTABILITY (relating to the accountability factor), and VAL-POSITIVE (the factor shows a positive effect). This method also supported iterative coding. Text fragments were often first coded using the very general code, in the example above DRL-CAPTURE. During later passes through the transcripts, text fragments coded with the DRL-CAPTURE could analysed more closely to determine which, if any, factors seemed to be at work and whether their effect was positive or negative.

A considerable amount of experimentation with the coding framework was undertaken at this stage of the analysis including assigning scales or values to codes and the use of multiple, more granular codes for a given text passage to support the use of logical operators in the Atlas/ti search mechanism. Also during this phase, both structural and directed networks were constructed from the within-case data. This involved building diagrams that express potential relations, both parent-child type structural and directed, potentially causal, between the codes.

The final step of this phase focussed on cross-case analysis. This involved reviewing the diagrams generated in the prior phases as overlays and identifying common structural and processional relationships. Huberman and Miles (1994) argue that qualitative research is especially well suited to identifying causal relationships, the how and why that emerge from the data, contrary to views from both the pure quantitative and purely constructivist schools. One of the most helpful techniques applied at this stage was to produce secondary documents that include all of the text passages for a given code from all of the interview transcripts. This supported 'drilling down' into a single concept to compare and contrast aspects and relations between the concepts across interview transcripts.

After the first six interviews had been analysed, a two-step analysis process was developed for coding the second round of interviews. This involved using a small set of general codes to mark text fragments as relating a high-level topic of interest. For example, a text fragment might be coded as DRL-CAPTURE (issues relating to capturing design rationale information). Once all of the transcripts had been coded at this level, all text fragment relating to a single high-level code were then exported to a single document, which was then used as a heterogeneous transcript in the next stage of analysis.

Text fragment size in the first stage of coding was intentionally large, sometimes a paragraph or two to include the entire statement relating to the high level code. This was to help ensure that important information was not left behind when the text fragment was exported to the new heterogenous transcript.

The following two tables show the final code sets used to analyse the two primary documents that emerged from the process described above. The first of these is Thesis Explanation Content.hpr, which consists of all fragments of interview transcripts relating to IS explanation content. The second is Thesis DR Focal.hpt, which consists of all fragments of interview transcripts relating to design rationale capture and use.

Table 12 - Explanation Content Codes

Codes	Codes
DR-EXP-CONTRADICTION	EXP-CONT-SCENARIOS
DR-EXP-INDIV-DIFFERENCES	EXP-CONT-SELF-DOCUMENT
DR-EXP-IS-DIFFERENCES	EXP-CONT-TACIT
DR-FOCAL	EXP-CONT-TESTING
DR-FOCAL-BRIDGE-INFO	EXP-CONT-TROUBLESHOOT
DR-FOCAL-NO-INTEREST	EXP-CONT-USER-MODEL
EXP-CONT-BUSINESS-PROCESS	EXP-CONT-VALUE
EXP-CONT-CONSTRAINTS	EXP-CONT-WHAT
EXP-CONT-CONTEXT	EXP-CONT-WHEN
EXP-CONT-CONTRAST	EXP-CONT-WHERE
EXP-CONT-CRITIC	EXP-CONT-WHO
EXP-CONT-DEFICIENCIES	EXP-CONT-WHY
EXP-CONT-EXCEPTIONS	EXP-CONTENT
EXP-CONT-FAQ	EXP-DELV-BALLOON
EXP-CONT-FUNCTIONAL	EXP-DELV-BOOKS
EXP-CONT-FUTURE-FEATURES	EXP-DELV-FAQ
EXP-CONT-HIGH-LEVEL	EXP-DELV-HELP-DESK
EXP-CONT-HOW	EXP-DELV-INTELLIGENT
EXP-CONT-HOW DOES IT WORK	EXP-DELV-PASSIVE
EXP-CONT-INCREMENTAL-DEVELOPMENT	EXP-DELV-PRINTED-MANUAL
EXP-CONT-INDIV-DIFFERENCES	EXP-DELV-TRAINING
EXP-CONT-META	EXP-DELV-WIZARDS
EXP-CONT-METAPHOR	FAC-CHAMPIONS
EXP-CONT-MOTIVATION	FAC-COMPETITION
EXP-CONT-NAVIGATION	FAC-DOC-VOLUME
EXP-CONT-NOT-CRITICISM	FAC-NO-ENJOYMENT
EXP-CONT-NOT-DANGER-FEATURES	FAC-NO-INTEREST
EXP-CONT-NOT-DELIBERATIONS	FAC-NO-TIME
EXP-CONT-NOT-DEPENDS	OUT-ACCOUNTABILITY
EXP-CONT-NOT-DEVELOPMENT	OUT-BREAKDOWNS

Codes	Codes
EXP-CONT-NOT-NOTHING	OUT-CHAMPIONS
EXP-CONT-NOT-SECURITY	OUT-CREDIBILITY
EXP-CONT-NOT-STAFFING	OUT-CUSTOMISATION
EXP-CONT-NOT-TECH	OUT-DAMAGE-CONTROL
EXP-CONT-NOT-USAGE-STATS	OUT-EFFECTIVE-USERS
EXP-CONT-NOT-WHY	OUT-PARTICIPATION
EXP-CONT-PLATFORM	OUT-REDUCE-SUPPORT
EXP-CONT-PREREQ	

Table 13 - Design Rationale Codes

Codes	Codes
A-CONTRADICTION	OUT-ALTERNATIVES
A-NEGATIVE	OUT-BETTER-COSTING
A-POSITIVE	OUT-BETTER-DECISIONS
FAC-ACCESS	OUT-BREAKDOWNS
FAC-APP-COMPLEXITY	OUT-BRIDGE-TECH-AND-USE
FAC-CAPTURE-TOOLS	OUT-COMMUNICATION
FAC-CHAMPIONS	OUT-COMPETITION
FAC-COMPETITION	OUT-CREDIBILITY
FAC-CORP-CULTURE	OUT-CUSTOMISATION
FAC-COST	OUT-DEVELOPER-TRAINING
FAC-DANGEROUS	OUT-DOCUMENTATION
FAC-DECREASE-METHOD-USE	OUT-EFFECTIVE-USE
FAC-INCREASED-TECH-KNOWLEDGE	OUT-LEGAL-PROTECTION
FAC-IS-IMPLICATIONS	OUT-MAINTENANCE
FAC-MAINTAINABILITY	OUT-NEW-USER-LEARNING
FAC-NO-CAPABILITY	OUT-OPEN-SOURCE
FAC-NO-INTEREST	OUT-ORG-MEMORY
FAC-NO-TIME	OUT-PARTICIPATION
FAC-PATIENCE	OUT-PARTICIPATION-ACCOUNTABILITY
FAC-TIME-TO-MARKET	OUT-PARTICIPATION-FEEDBACK
FAC-TOO-MANY-COOKS	OUT-RELATE-IS-BUS-PROCESS
FAC-TOO-MUCH	OUT-REQUIREMENTS-TRACING
FAC-TOOLS	OUT-SUPPORT
FAC-TRANSLATION	OUT-TRAINING
INFO-RATIONALE	OUT-USER-COMFORT

Appendix D – Study 1 Coded Data Sources

The following table lists the major conceptual codes reported in Chapter 5 along with the Atlas/ti files that are the source code counts and text quotations provided.

Table 14 - Coded Data and Atlas/ti Source Files

Code	Source
Summary of Findings Related to Explanation Content	
Operational Explanations	
How do I use it?	Thesis Explanation Content.hpr
What does it do?	Thesis Explanation Content.hpr
How does it work?	Thesis Explanation Content.hpr
Why Explanations	
D-N Explanation	Thesis Explanation Content.hpr
Pragmatic Explanation	Thesis Explanation Content.hpr
Functional Explanation	Thesis Explanation Content.hpr
Rational Choice Explanation	Explanation Search.hpr
Summary Factors and Effects of DR Explanations	
Factors	
Lack of interest	Thesis DR Focal.hpr
Champions	Thesis DR Focal.hpr
Lack of Time	Thesis DR Focal.hpr
Design by Committee	Thesis DR Focal.hpr
Information Translation	Thesis DR Focal.hpr
Competition	Thesis DR Focal.hpr
Outcomes	
Participation	Thesis DR Focal.hpr
Communication	Thesis DR Focal.hpr
Organisational Memory	Thesis DR Focal.hpr
Credibility	Thesis DR Focal.hpr
Design Decisions	Thesis DR Focal.hpr

Code	Source
Customisation	Thesis DR Focal.hpr
Requirements Tracing	

Appendix E – Study 1 Coded Transcript Samples

The following pages provide samples showing how codes were applied to interview transcripts. Two samples are provided, the first shows all of the text fragments coded for the explanation content type for D-N explanations EXP-CONTENT-CONSTRAINTS and the second for design rationale capture and use factor FAC-CHAMPIONS. Transcripts have been edited to remove references to the interview participants responsible for the different quotations.

EXP-CONTENT-CONSTRAINTS

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-HOW DOES IT WORK] [EXP-CONT-WHAT]

I think just a full textual explanation of what that term is or what the impact of putting a value in or what the allowed values are and that.

Codes: [EXP-CONT-CONSTRAINTS]
What amount of up time has that system got? How much access to I have to that system? What are the restrictions on the use of that system? All the usual what I would term administrative type elements associated with it. And also what my role is as a user of that system in the form of either data integrity and any other legal requirements that might come along.

Codes: [EXP-CONT-CONSTRAINTS]

Right. So examples of what will happen when they do a certain, perform a certain task.

Correct, yep. I think that's mostly it. You know if there's performance implications, if you have to wait for any type of interaction to occur, you'd probably want that to be involved as well so you give them a reasonable expectation for response times and those sorts of things. Generally that is what I would expect.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-HOW]

Where people were working within a set of constraints like they knew damn well what their front end tool was. That's changing now of course. We know what the front end tools are, they're the Cognos front end tools. So now we can start applying more cookbook approach type documentation sort of thing.

Codes: [EXP-CONT-CONSTRAINTS]

Right, okay. Well that takes care of the next question. Do you think that there's any way that the information that's generated during analysis and design can be transformed to make it more useful to end users or customers?

Well, yeah, I think that the analysis and design aspect of the project obviously starts with you know a specification of the objectives and the constraints on the project, okay? That's kind of what bounds the project and stating that clearly as part of the overview in the end users documentation could be helpful, saying this product is designed to do X, Y and Z and not A, B and C because of C,D and E. It could give a perspective on what's happening.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-CONTEXT] [EXP-CONT-MOTIVATION]

Well, anytime you have a design team involved, you have to have you know a system in place that you know gives them all the same context of decision making so they can you know they can both participate in and support the decisions. So this is, what I understand this is basically is a way of communicating to people what our you know our objectives are and what our approach is and why we're taking a specific approach. And like here, I mean, if we're building a product that's based on the Mac instead of the PC, all the PC you know bigots will be upset about that unless you can explain to them, we've done this analysis, cost is the key objective, life cycle cost of the Mac is 20% lower than the PC, that's the facts of life guys. I mean you get by in, but if you basically don't give that context in a way where they can understand it, then they won't support it.

Codes: [EXP-CONT-CONSTRAINTS]

Well, there's constraints at the functional level too but I mean for instance with a railroad. You want to keep two trains from running into each other. I mean that's basically on of the

Codes: [EXP-CONT-CONSTRAINTS]

Well graphics are very useful, of course. Showing people picture of what it is they're going to be seeing, every step by the way. I suppose screen shots is what I'm going for here. I do a lot of technical editing and sometimes on topics I know nothing about, I'll give you an example. A guy wrote a book about special effects for movies and the way to create these with some kind of computer programme I had never heard of. Reading his book as he wrote it, I wouldn't have had a clue what to do with this even if I had ever used this programme, he never once said, launch it or what is the platform requirement or anything.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-FUNCTIONAL] [EXP-CONT-NOT-DEVELOPMENT] [EXP-CONT-WHEN]

In terms of analysis and design, if you mean like for example, we come up to a problem and you know here are the variables, here are all the decision variables and here's a search space that this set of decision variables creates and here's a size and you know how many iterations do we need to go through to get a good coverage in a system. That kind of detail, the end user's not interested in. Okay. The kind of information that they're interested in that you know, how does it take for the system, for the software to arrive at a solution? What kinds of constraints can it represent? So they're more interested in terms of you know functionality and capability as opposed to how we arrive at those functionalities and capabilities.

Codes: [EXP-CONT-CONSTRAINTS]

Right. That is a very part of what we call the constraint model. Because a constraint with no explanation really does not provide as much value as it can, in terms of confidence, in terms of knowing what's going on in the system.

Codes: [EXP-CONT-CONSTRAINTS]

Yes. I think so. I think this explanation capability still holds its value even at the simple end of the of the spectrum. In my experience with the software that I work with, the same software framework is used for simpler kinds of problems where you only have you know five or six constraints, okay, all the way to very complicated system where you have hundreds of constraints, okay. So our system, the software framework does not change. The same optimisation techniques use the same constraint modelling language is used both for the simpler system with a fewer number of constraints as for the more complicated system with hundreds of constraints. And we find that the explanation capability is still very valuable even in the case where you only have five or six constraints.

Codes: [EXP-CONT-CONSTRAINTS]
.....the benchmarks for time were just driven what we knew about how long people were willing to wait for things to work before they got pissed off. Cause the worst thing that can happen is something takes so long that the user then touches something else and then in effect the system is struggling to keep up with the user's expectations and it's total chaos. So an explanation of why we had the kind of run time features that we did would make reference to the benchmarks.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-SCENARIOS] [EXP-CONT-TESTING]

And explanation of why we had the user interface that we did would make mention of the actual user interface testing that we've done. An explanation of the developer framework, should make reference to observed and actual instances of the use of the developer platform. So actual code that we'd seen written or developed. So those would be those kinds of constraints.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-WHY]

We had one other constraint which was crucial which was we were a highly resourced constrained platform. So we had like four megs of RAM and a four meg ROM and that was it. And we didn't have a lightening fast processor either, although for the second generation we probably had more processor than we'd ever had cause we had a MIPS R3000 derivative. But so I would have thought an explanation should have made reference to the reason why the feature was there.

Codes: [EXP-CONT-CONSTRAINTS]

And there we'd site the form factor as our explanation. I should say and that was the other thing is, hey, we're a small device, we're battery powered, it's a touch screen, it's a half VGA which was enormous by the standards of that day, the Pilot's tiny. So we'd make reference to the form factor. I don't know if you're interested, no, I was going to mention

Codes: [EXP-CONT-CONSTRAINTS]

We support Java Script to the extent that we can with these constraints mainly don't have a mouse, don't have this, don't have that. And generally people would get it because if you just say no, they'd put you know, incomplete Java Script. You have to say well the reason we don't do it, there's no way to do it. And there is became important so they didn't perceive it as a limitation of the system, they perceived it I'd like to think as a beneficial adaptation.

Codes: [EXP-CONT-CONSTRAINTS]

Well, we're a constrained platform, if we had all the kinds of virtual tables that you've got to have with C++, we'd chew up RAM like you can't believe. Also we needed an object model that could do what we call shadowing. So you could have an object in ROM and then you'd just make a copy of that object. And we had to make sure that our object hierarchies were sufficiently granular that we only pulled out as much inter-RAM as necessary. So these are sorts of thing a developer, a developer platform for our customers, we responsive to and they would understand those kinds of things

Codes: [EXP-CONT-CONSTRAINTS]

So yeah, those people do need to be understood, given those constraints, particularly when you're dealing with a new class of device, which we were. A very small hand-held devices. A lot of people hadn't thought through the implications of being intermittently connected, having a highly constrained RAM budget, power budget. Those sorts of things became very important. So yes, you do have to give that kind of information. And a lot of that is just straightforward explanation. You say, small device, you can't have this, small device, you can't have that, small amount of RAM, can't do that.

Codes: [EXP-CONT-CONSTRAINTS]

Constraints was a big thing that we would explain. In fact, as I just mentioned I guess, two our of three of my examples were a case of explaining the constraints on the design. It's sort of the opposite of, as I said earlier, software can do anything. Yeah, but it's on hardware.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-HIGH-LEVEL]

I think broad overall descriptions on the level of we always have this problem and so here's how we got round it, can contribute because it makes the customer understand what it is they're buying. So that they, they don't necessarily know what the problems are and why something should be easy or difficult. And so they don't necessarily know why one product is better than another or whether it is. And so giving them some broad understanding of the sort of technical issues that come up with this type of product, is beneficial because they see why, why ours is potentially better.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-WHY]

Cause they don't have to sit there wondering why on earth the product has to do something, why can't it do something else? So they understand that there's, that there are, there are technical issues which any company has to address when it produces a product of that sort.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-PREREQ]

So if it's a system for instance it may have a chapter on what the customer has to facilitate to make the system work, i.e., the system will get plugged into their corporate wide area networks so it would document what kind of band width we need to make the system perform as required.

Codes: [EXP-CONT-CONSTRAINTS]

In other, the only other instance I can think where it actually helps explain this, is an end user asks for an enhancement, sometimes you have to explain to them, well if it's an easy enhancement and we can go out and say yeah, that's really easy, it'll cost this small amount of money, they don't care how the design works. But if we go out to them and say well that's actually quite difficult or that's actually impossible but this alternative is, they always ask why. And it's at that point some things you actually have to start explaining, again at the conceptual level or layman's level how the system is put together, to explain why that enhancement would actually be difficult or impossible, because the design of the software behind the current functionality, would be very difficult to change to provide that enhancement.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-CONTRAST]

And you're actually again explaining, you're right, explaining the rationale between, to say cause everything's a compromise. You know every piece of user functionality

can't be perfect every time you know. Software isn't magic. You have to make some compromises. You have to say, well the critical part of the system has to do this and you have, at the end of the day, the system has to be at its best in these circumstances. Like if there's a storm. In our industry, if there's a lightening storm and the system is under a lot of strain because we get lots of alarms then, and there's lots of customers they're supplying and the control engineers are going crazy, at that point the system has to perform at its best. And the system has to be able to process large amounts of data as fast as possible. That actually means that when you're loading the system for the very first time, when you just bought it, it's not as fast to load it with data. Everybody says, it's not very good at loading data, at creating new objects in the database. It's kind of slow when it does that. Can you not make that faster? And we basically have to say well the entire design of the system is designed that data changes incredibly fast, data creation transactions because of that are a lot slower. And it's much better to have a data capture, a data loading phase in your project when you're moving towards a live system that's four months instead of two months rather than have a slow system in every storm in every winter you have from now until you retire the system. So there's things like that you have to explain.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-PREREQ]

Well what jumps out, what does jump out at me is how the resources that the application is using. So you can often go a number of steps down a road and find you haven't got enough disk space and it's something you should know up front. Running out of memory, very poorly managed and that may be an OS thing. As a user I've got to the put I've got to cut and paste and the whole thing's bombed. Or I don't have enough memory to complete the operation. And maybe that doesn't really mean anything to me as a user but I should know it before I embark.

Codes: [EXP-CONT-CONSTRAINTS] [EXP-CONT-HOW DOES IT WORK]

Okay, okay. Do you think that information from the analysis and design phase can contribute to an explanation of a system feature or process?

Absolutely. And that's again, now you are at the architecture level, and when I keep saying that I mean when the system, the main packages, components, pieces, logical boundaries, interfaces, how they talk, will certainly help you describe what the system's doing especially if it's complex, if it's distributed, if it has many parts.

FAC-CHAMPIONS

Codes: [A-POSITIVE] [FAC-CHAMPIONS]

Yeah, I don't think so. Again I'm sure there's a small

sub-set of users who would understand the architectural aspects and the other elements. I don't see the benefit of it.

Codes: [A-NEGATIVE] [FAC-CHAMPIONS] [FAC-NO-INTEREST]

Um, yeah, I actually think there's plenty of stuff that they don't necessarily need to see. I think you know larger applications that are highly distributed, they have very evolved architectures, that you know have lots and lots of intricate detail behind them, is almost meaningless to an end user and it ultimately doesn't affect their usage of the system. I mean if they've got the band width to understand it and they're the inquisitive types that just sort of want to know, then maybe there's an impact there.

Codes: [A-NEGATIVE] [A-POSITIVE] [FAC-CHAMPIONS] [FAC-NO-INTEREST]

Right. And why do you think that's important, providing that sort of information?

Well, I mean there's a certain class of user who probably is curious as to why specific decisions were made in a product particularly if the user tends to relate to that product. I mean it's an important part of, if he has a very emotional life so to speak, they might want to know the background. But in general I think people are you know concerned primarily about form, fit and function, you know, as opposed to how you got there.

Codes: [FAC-CHAMPIONS]

So by the time we cut over and go live, there's the core part of the business people in the project team who are very knowledgeable about the system. And have been through a lot of the project will have access to all the project documentation, will have access to any system generated help facilities

Codes: [A-POSITIVE] [FAC-CHAMPIONS] [OUT-BETTER-DECISIONS]
[OUT-PARTICIPATION-FEEDBACK]

But if the end user is let's say you know like you know engineers and you know those kind of people, then yeah, I would, I can see them you know having more interest in this kind of information. They may be able to give better information to the software developers in terms of you know, you don't understand this right? You know it's not really this way, it's this way. Your picture's all wrong. Whereas the customers that I've worked with they know their job, they don't really know the arguments that we go through in making design choices. I work with them more as a you know real world check, are we doing the right thing? But really after we've done all the design analysis and design choices.

Codes: [A-NEGATIVE] [A-POSITIVE] [FAC-CHAMPIONS]

But again I think that you know you need to ask people who work with more the you know, more stable products you know. The ones that are already well established you know. Do

they see any harm in it? In my customer space, no. My development partners, I don't see any harm in allowing them to look at this. Do I think that they will use it? I don't think they'll use it. Everyone except the champion, right? The champion may use it as you said.

Codes: [A-POSITIVE] [FAC-CHAMPIONS] [FAC-COST]

It's overly dependent on gurus but this is, this is a bit speculative on my part after, after ten months of being there. That's just my feeling. Of course what I'm suggesting using more documentation without quite a large cost in hours of work, distracting people from getting on with developing the product.

Codes: [A-POSITIVE] [FAC-CHAMPIONS] [OUT-ORG-MEMORY]

I mean the way that information is spread in the group, is that there are gurus and there are other people deliberate, time is set aside for that guru to transfer the knowledge to another person so in an attempt to build up a knowledge base of excellence of different people. And so there will be maybe the guru who knows all about testing and the application and then they will pass that along to one other person. And if that guru was ever to leave, it would then have to be passed on to another second person.

Codes: [A-POSITIVE] [FAC-CHAMPIONS]

Because at the end of the day what we found in the past is there is nothing better than actually getting two hours of the guy who knows it inside out. If you get two hours of his time in a closed office with a flip chart, there's not, there isn't much better than that to actually get what we need.

Codes: [A-NEGATIVE] [A-POSITIVE] [FAC-CHAMPIONS]

Okay. Do you think that users are interested in the different design alternatives that are considered in a design process and why certain ones are chosen over others?

Codes: [FAC-CHAMPIONS]
Right, okay. Do you think that, do you think that users are interested in design alternatives that are considered early on in a project and why certain ones are accepted or implemented and why certain projects are rejected?
I think some users are. A lot of our users are systems literate and they may have a passing interest in the whys but again it's just a small minority I would say who have the time really to concern themselves with those sort of issues.

Codes: [A-POSITIVE] [FAC-CHAMPIONS] [OUT-COMMUNICATION]

Okay. Do your users have access to information that's produced during the analysis and design phases? So do your users look at design documents and analysis documents?

Yes, some of them do. Typically the way it'll work here is you'll have a team that's responsible for producing this

high level design document and they'll be some key users that have been designated by the business community as subject matter experts in those particular fields. And they'll be the ones that are responsible for providing the requirements and in fact the signing off requirements documents. So those particular individuals will typically see all the documents. Now the broad user community typically will not and the assumption is that these subject matter experts are representative of the user community. Now that's often a tricky assumption because sometimes they're not. So it's important from the business community side to make sure that if they're going to designate a team of people that the IT are going to refer to as subject matter experts, that they have a good cross-section and make sure that all the key interest groups from the business community are represented and represented properly. Now whether they actually really do that effectively or not you know is you know is neither here nor there. But that's typically the way it should work or the way it works here anyway.

Codes: [A-NEGATIVE] [A-POSITIVE] [FAC-CHAMPIONS] [FAC-TOO-MANY-COOKS]
[OUT-CREDIBILITY]

So my answer would be similar in that I think the, in actually making that decision, I think that the representative, the designated representatives of the user community who have design authority, should be involved in actually making that decision along with you know the design or development teams. And they should be made aware of all the different options that are being considered. And the pros and cons of each and in terms not only of how it affects the system, but what it costs. And those decisions should be properly made, involving those representatives of the business community. Subsequently whether it's going to be, help anybody to see you know what other options were there and you know I think that it might, it might help. I mean again I mean to know why decisions were made I suppose would answer a lot of questions. If they say well why doesn't the system do this way and then they could say well they made this decision and these were the reasons, well at least it will answer their questions. But they might not like it. I think it would probably be you know be useful to them, yes. I don't think it's necessarily useful from the perspective of an application development team again because you've got this danger of keeping on reopening the same can or worms over and over again and keep on having to revisit design decisions that have been made and reopen them. And really you want to avoid doing that but I could understand why the user community would want to have that information.

Codes: [A-POSITIVE] [FAC-CHAMPIONS] [OUT-BETTER-DECISIONS]

But what I've found in a lot of cases is you know we'll make, you know you might have this planning team and you'll come up with this design, and you'll make all these decisions and you know you'll sit down with the business analyst and you'll set down with the end users and you'll say, there's all these different ways we could do this and

you know we've got to reconcile you know, the revenue services people need to have this information and I now that the order entry people don't want to provide it but they've got to. And the order entry people want to have the ability to do their jobs really quickly and revenue services don't really want them to do that. They want to have all these audit controls. So in the end a lot of these decisions are very much compromise decisions you know where almost they're decisions that are made that don't really made anybody ecstatically happy but everybody's kind of grudgingly willing to accept because anything else either just suits one party and not the other or vice versa. So often a lot of these decisions are compromise decisions and when you make them, often you'll move forward, and several steps later you'll realise that you just made a big mistake. You know you'll think well you know we were trying to reconcile all these requirements and we thought this was the best way to do it but now we just realised that that it just flat out doesn't work like this. It you know it takes too long to run and it's unwieldy and you know we need to change our minds. And but now we can't because we're going to implement in three weeks and we made this decision so now we've got to we've got to like quickly change the way we're doing it and go some other way purely out of expedience more than because we think it's a very clearly well thought out design. And a lot of our design, ultimately and being pragmatic, the end system when it's actually delivered, is you know, is often ends up being a cobbled together set of compromises and last minutes changes to fix things that, you know, and whether the end system that's develops you know really matches what it was, what you thought it was going to be when you documented your design, often it doesn't. And so often you know, so the result of that would be of course if the user you know wanted a clear explanation of why the system was doing what it was doing, he might actually not be very happy with the information he got. So there's an element of but I mean that's just, I mean that's not good and that's not an argument against doing it. I mean maybe that's all the more reason for doing it. You know if you've got like a flaw in the system, well you know it's not going to do you much good in the long term to hide it. But and so yeah, those are the kind of comments that spring to mind. I think that ultimately my guess is that it would help, it would be a great help to some members of the user communities but maybe not so much to others depending on what the system was doing and what their role was in it.

Codes: [A-POSITIVE] [FAC-CHAMPIONS]

Right. So do you think that that information, the analysis and design information, should be made available to end users?

Yeah I think so, if they want it. Definitely, I mean you know there's no harm in somebody you know having as complete of an understanding as possible on you know on a new system. I mean especially if somebody is really showing an interest it could you know lead to a new role for them in the future.

Codes: [A-POSITIVE] [FAC-CHAMPIONS]

No we, not end users. I always used user champions if I can. So on a big project somebody will be nominated as a champion for users. So there's people do pilot groups, focus groups, user champions, whatever. I haven't actually provided them with analysis documents, no.

Codes: [A-NEGATIVE] [FAC-CHAMPIONS] [FAC-COMPETITION]

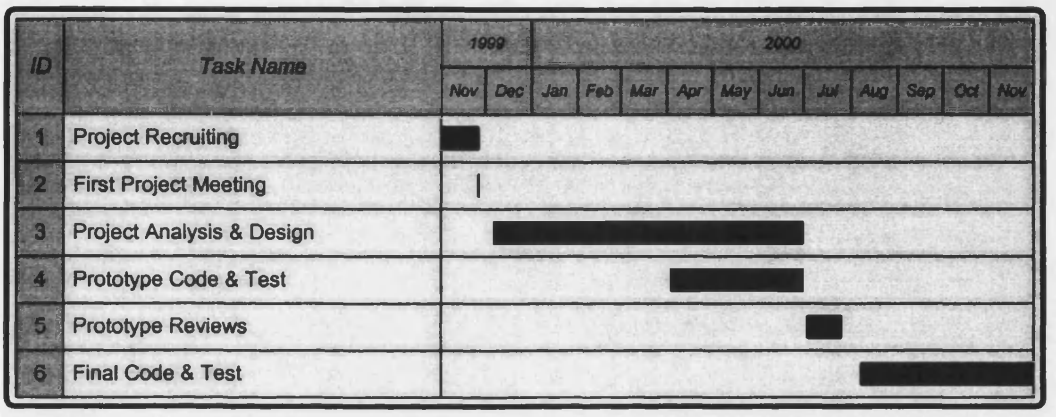
You don't think that that sort of information is or do you think that that information would contribute to usability?
To a certain small section of the users possibly so.
And who would they be?
The serious _____. The
people who probably who have tried writing their own code to solve a problem.
So do you think that would be a
threat then to give that sort of information out? RG: Yeah, I wouldn't want to give that information out.

Appendix F – Study 2 Project Team

The following table lists the members of the Study 2 project team (see Chapter

Name	Core Team	Development Experience
Ali Al-Amudi	√	√
Lucinda Chen	√	
Odysseas Dotsikas	√	
Steven Haynes	√	√
Ashutosh Khanna		
Dicle Kortantamer	√	√
Harry Mann	√	
George Saliaris-Fasseas	√	
Soon Shean Ong		
Noelle Siong	√	√
Nina Sukhabhai	√	
Paul Vant		
Simon Villamayor		

Appendix G – Study 2 Project Timeline



Appendix H – Study 2 QOC (raw)

The following raw listing is the raw-format QOC from the VentureQuery case study (Study 2) reported in Chapter 6.

Project ID: 3 Name: Venture Query

Outline: 141 Number: 01.0 Name: Application

Question: 21 Name: What is the appropriate system model metaphor? Parent Option: 0
- Option: 16 Name: A board game which is played Score: 0 Selected? true
  - Criterion: 108 Name: Represent progress
  - Metacriterion: 45 Name: Easy to use questionnaire (respondent)
  - Term: Score: 0
  - Criterion: 111 Name: Ease of implementation
  - Metacriterion: 31 Name: Simplicity of construction
  - Term: Score: 0
  - Criterion: 113 Name: Uniqueness
  - Metacriterion: 19 Name: Make the system fun to use
  - Term: Score: 0
- Option: 17 Name: A business simulation Score: 0 Selected? false
  - Criterion: 108 Name: Represent progress
  - Metacriterion: 45 Name: Easy to use questionnaire (respondent)
  - Term: Score: 0
  - Criterion: 111 Name: Ease of implementation
  - Metacriterion: 31 Name: Simplicity of construction
  - Term: Score: 0
  - Criterion: 113 Name: Uniqueness
  - Metacriterion: 19 Name: Make the system fun to use
  - Term: Score: 0
- Option: 222 Name: A ladder to climb Score: 0 Selected? false
  - Criterion: 108 Name: Represent progress
  - Metacriterion: 45 Name: Easy to use questionnaire (respondent)
  - Term: + Score: 0
  - Criterion: 111 Name: Ease of implementation
  - Metacriterion: 31 Name: Simplicity of construction
  - Term: Score: 0
  - Criterion: 113 Name: Uniqueness
  - Metacriterion: 19 Name: Make the system fun to use
  - Term: Score: 0
- Option: 223 Name: Moving through a building/rooms Score: 0 Selected? false
  - Criterion: 108 Name: Represent progress
  - Metacriterion: 45 Name: Easy to use questionnaire (respondent)
  - Term: Score: 0
  - Criterion: 111 Name: Ease of implementation
  - Metacriterion: 31 Name: Simplicity of construction
  - Term: Score: 0
  - Criterion: 113 Name: Uniqueness
  - Metacriterion: 19 Name: Make the system fun to use
  - Term: Score: 0
- Option: 224 Name: A computer chip (circuit) Score: 0 Selected? false
  - Criterion: 108 Name: Represent progress
  - Metacriterion: 45 Name: Easy to use questionnaire (respondent)
  - Term: Score: 0
  - Criterion: 111 Name: Ease of implementation
  - Metacriterion: 31 Name: Simplicity of construction
  - Term: Score: 0
  - Criterion: 113 Name: Uniqueness
  - Metacriterion: 19 Name: Make the system fun to use
  - Term: Score: 0

Question: 33 Name: What are the components of the game? Parent Option: 16
- Option: 33 Name: Squares Score: 0 Selected? false
  - Criterion: 36 Name: Preserve the system metaphor

Metacriterion: 16 Name: Conform to the system metaphor
Term: + Score: 0
Option: 379 Name: Projects Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0
Option: 380 Name: Square Sections Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0
Option: 381 Name: Questions Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0
Option: 382 Name: Answers Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0
Option: 383 Name: Metrics Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0
Option: 384 Name: Sessions Score: 0 Selected? false
Criterion: 36 Name: Preserve the system metaphor
Metacriterion: 16 Name: Conform to the system metaphor
Term: Score: 0

Question: 34 Name: How will the system flow work? Parent Option: 0
Option: 34 Name: Only support a full 'play' Score: 0 Selected? false
Criterion: 37 Name: Simpler design and construction
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 38 Name: More flexible user interface
Metacriterion: 30 Name: Flexibility of use
Term: - Score: 0
Option: 35 Name: Partial play supported Score: 0 Selected? false
Criterion: 37 Name: Simpler design and construction
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 38 Name: More flexible user interface
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0
Option: 72 Name: Ask a Question, get an Answer, Apply a Metric, give a Score Score: 0 Selected? false
Criterion: 37 Name: Simpler design and construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 38 Name: More flexible user interface
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Question: 40 Name: What is the purpose of the system? Parent Option: 0
Option: 41 Name: Gain experience in software development Score: 0 Selected? false
Option: 42 Name: Profit Score: 0 Selected? false
Option: 43 Name: Test bed for a research project Score: 0 Selected? false
Option: 343 Name: An educational tool Score: 0 Selected? false

Question: 41 Name: Will any research apparatus be built into the software? Parent Option: 43

Question: 44 Name: What is the system platform? Parent Option: 0
Option: 47 Name: System will be web-based (cross-platform) Score: 0 Selected? false

Question: 45 Name: How will the game be scored? Parent Option: 38
Option: 49 Name: Metrics class evaluates User Answer Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0

Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 67 Name: One score for each sub-section Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 68 Name: One score for the entire game Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 70 Name: Provide URLs in addition to score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 73 Name: Single quantitative score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction

Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 74 Name: Percentage score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 75 Name: Qualitative score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 188 Name: Use a normalised, ordinal scale Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 281 Name: Default by the entire play, but let them drill down to an individual score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0

Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Option: 282 Name: A weighted score for each Questions Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Option: 283 Name: Separate window displays score on demand Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Option: 284 Name: A graphical display of the score Score: 0 Selected? false
Criterion: 49 Name: Present information in digestible chunks
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 50 Name: Maintain user interest through the game
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Criterion: 51 Name: Present useful information
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 79 Name: Ease of construction
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 132 Name: Support partial play
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 133 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Question: 51 Name: What are the system classes? Parent Option: 0
Option: 58 Name: User Class Score: 0 Selected? false
Option: 59 Name: Square Class Score: 0 Selected? false
Option: 60 Name: Score Class Score: 0 Selected? false
Option: 61 Name: Question Class Score: 0 Selected? false
Option: 62 Name: Answer Class Score: 0 Selected? false
Option: 63 Name: Help Class Score: 0 Selected? false
Option: 64 Name: Dependency Class Score: 0 Selected? false

Option: 76 Name: URL Score: 0 Selected? false
Option: 81 Name: Session Class Score: 0 Selected? false
Option: 82 Name: Advice Class Score: 0 Selected? false
Option: 110 Name: Path Master Class Score: 0 Selected? false
Option: 111 Name: User Square Path Score: 0 Selected? false
Option: 112 Name: Square Section Class Score: 0 Selected? false
Option: 131 Name: System Defaults Class Score: 0 Selected? false

Question: 62 Name: How does the application time out? Parent Option: 0
Option: 90 Name: Just time out after some period of inactivity Score: 0 Selected? false

Question: 67 Name: Do we track how/whether users use advice and links? Parent Option: 0
Option: 97 Name: Track users navigation through the system Score: 0 Selected? false

Question: 68 Name: How will we track system navigation Parent Option: 97

Question: 73 Name: How is the game initially created? Parent Option: 0

Question: 100 Name: Should some Sections, Squares, Questions be flagged as introductory? Parent Option: 0
Option: 155 Name: Yes Score: 0 Selected? false
Criterion: 69 Name: Allows us to create the User Square Path from initial Questions
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Option: 160 Name: No Score: 0 Selected? false
Criterion: 69 Name: Allows us to create the User Square Path from initial Questions
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0

Question: 115 Name: Create a separate module for Question dependencies? Parent Option: 0

Question: 122 Name: How many pages do we need? Parent Option: 0
Option: 202 Name: Home page Score: 0 Selected? false
Option: 203 Name: Introduction page Score: 0 Selected? false
Option: 204 Name: About the Venture Query project Score: 0 Selected? false
Option: 205 Name: User registration page Score: 0 Selected? false
Option: 206 Name: Resource page Score: 0 Selected? false

Question: 125 Name: Do we allow preview of questions with a full play? Parent Option: 0

Question: 126 Name: What is on the first page? Parent Option: 0
Option: 216 Name: Link: About Us Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 217 Name: Preview Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 218 Name: Questionnaire Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 219 Name: Contacts Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 220 Name: Feedback Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 233 Name: Paragraph describing project Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 234 Name: Resources Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 235 Name: How questionnaire evaluated Score: 0 Selected? false
Criterion: 118 Name: Simple
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0

Question: 138 Name: What is the application architecture? Parent Option: 0
Option: 256 Name: Applet Score: 0 Selected? false
Criterion: 127 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 257 Name: Servlet Score: 0 Selected? false
Criterion: 127 Name: Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: + Score: 0

Question: 180 Name: Include debug mode for viewing completed questionnaires? Parent Option: 0

Question: 192 Name: Provide advice in addition to the score? Parent Option: 75

Outline: 128 Number: 02.0 Name: System Defaults

Question: 86 Name: What does the System Defaults Class do Parent Option: 131
Option: 132 Name: Store the First Square ID Score: 0 Selected? false

Question: 166 Name: What are the attributes of Project Defaults? Parent Option: 0
Option: 333 Name: Questionnaire Home Page Score: 0 Selected? false
Option: 335 Name: Graphics URL Score: 0 Selected? false
Option: 336 Name: Base URL Score: 0 Selected? false
Option: 337 Name: Questionnaire (small) logo file name Score: 0 Selected? false
Option: 338 Name: Help URL Score: 0 Selected? false
Option: 367 Name: First Square ID Score: 0 Selected? false
Option: 368 Name: Allowed Answer Input Types Score: 0 Selected? false
Option: 370 Name: Introduction (large) logo file name Score: 0 Selected? false
Option: 371 Name: Introduction Message Score: 0 Selected? false

Question: 172 Name: What is the purpose of the base and other URLs? Parent Option: 336
Option: 344 Name: Allow questionnaire builders to host their own page materials Score: 0 Selected? false
Criterion: 153 Name: More difficult to build questionnaires
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 154 Name: More flexible, customisable questionnaires
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 345 Name: Ease administration of the actual VQ application Score: 0 Selected? false
Criterion: 153 Name: More difficult to build questionnaires
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 154 Name: More flexible, customisable questionnaires
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Outline: 95 Number: 03.0 Name: User

Question: 49 Name: Keep history of the user? Parent Option: 0

Question: 82 Name: What information do we need from the user at login Parent Option: 0
Option: 121 Name: User name Score: 0 Selected? false
Criterion: 57 Name: Preserve user privacy
Metacriterion: 28 Name: Protect user privacy
Term: - Score: 0
Option: 122 Name: User email Score: 0 Selected? false
Criterion: 57 Name: Preserve user privacy
Metacriterion: 28 Name: Protect user privacy
Term: - Score: 0

Question: 83 Name: How will user information be used Parent Option: 122
Option: 123 Name: Deduplicate users Score: 0 Selected? false

Question: 135 Name: What is the relationship of User to Session Parent Option: 0
Option: 248 Name: One User, many Sessions Score: 0 Selected? false
Criterion: 126 Name: Help track partial plays by User
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0

Question: 139 Name: What does the User class do? Parent Option: 0
Option: 258 Name: Create the Session Score: 0 Selected? false
Option: 261 Name: Display the Welcome message Score: 0 Selected? false

Question: 171 Name: What are the attributes of the User class? Parent Option: 0
Option: 342 Name: User email Score: 0 Selected? false

Outline: 125 Number: 04.0 Name: Session

Question: 37 Name: Will the game be scored? Parent Option: 16
Option: 38 Name: Score the game numerically Score: 0 Selected? true
Criterion: 40 Name: Make the system fun to use
Metacriterion: 19 Name: Make the system fun to use
Term: Score: 0
Criterion: 41 Name: Make the system easy to build
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 42 Name: Make the results easy to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 75 Name: Scoring must be accurate to be useful
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 39 Name: Provide qualitative advice Score: 0 Selected? true
Criterion: 40 Name: Make the system fun to use
Metacriterion: 19 Name: Make the system fun to use
Term: Score: 0
Criterion: 41 Name: Make the system easy to build
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 42 Name: Make the results easy to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 75 Name: Scoring must be accurate to be useful
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 96 Name: Questions themselves are enough advice Score: 0 Selected? false
Criterion: 40 Name: Make the system fun to use
Metacriterion: 19 Name: Make the system fun to use
Term: Score: 0
Criterion: 41 Name: Make the system easy to build
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 42 Name: Make the results easy to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 75 Name: Scoring must be accurate to be useful
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 181 Name: Do not score the game Score: 0 Selected? false
Criterion: 40 Name: Make the system fun to use
Metacriterion: 19 Name: Make the system fun to use
Term: Score: 0
Criterion: 41 Name: Make the system easy to build
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 42 Name: Make the results easy to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 75 Name: Scoring must be accurate to be useful
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0

Question: 61 Name: How to calculate score with skipped Squares? Parent Option: 68
Option: 89 Name: Score are calculateded based on the % of completed Squares Score: 0 Selected? false

Question: 63 Name: What does the Session class do? Parent Option: 81
Option: 91 Name: Manage a play Score: 0 Selected? false
Option: 92 Name: Manage time out Score: 0 Selected? false
Option: 93 Name: Identify the user Score: 0 Selected? false
Option: 246 Name: Create the user square path when instantiated Score: 0 Selected? false
Option: 247 Name: Create the user object Score: 0 Selected? false
Option: 253 Name: Identify the first question (or square) Score: 0 Selected? false
Option: 262 Name: Display the Introduction page Score: 0 Selected? false
Option: 263 Name: Call the first Square Score: 0 Selected? false
Option: 321 Name: Display the end message Score: 0 Selected? false

Question: 64 Name: Do we capture the user email in the Session Parent Option: 93
Option: 94 Name: Capture the user email Score: 0 Selected? false

Option: 341 Name: Do not capture user email Score: 0 Selected? false

Question: 77 Name: What does the Session class do Parent Option: 81
Option: 107 Name: Tracks the time and date of the Session Score: 0 Selected? false
Option: 108 Name: Calculates the total duration of the Session Score: 0 Selected? false
Option: 109 Name: Calculates the total score for the session Score: 0 Selected? false
Option: 264 Name: Get the first Square Score: 0 Selected? false
Option: 290 Name: Presents the welcome page and text Score: 0 Selected? false
Option: 313 Name: Randomizes the user into a group Score: 0 Selected? false
Option: 316 Name: Constructs the user square path Score: 0 Selected? false

Question: 81 Name: How does a Session begin Parent Option: 0
Option: 119 Name: User logs in Score: 0 Selected? false
Option: 120 Name: During login, a Session is created Score: 0 Selected? false
Option: 124 Name: During login, a User object is created Score: 0 Selected? false
Option: 125 Name: Login the Session commence data and time Score: 0 Selected? false
Option: 126 Name: Initialise the Score object Score: 0 Selected? false
Option: 127 Name: Create the User Square Path object Score: 0 Selected? false
Option: 245 Name: From home page, user clicks on Simulation link Score: 0 Selected? false

Question: 117 Name: How will the score be displayed to the user? Parent Option: 0
Option: 191 Name: At the end of the Session Score: 0 Selected? false
Criterion: 81 Name: Simplify the main question squares
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0
Criterion: 82 Name: Hold the user's interest through the questionnaire
Metacriterion: 42 Name: Make the game interesting
Term: - Score: 0
Option: 192 Name: Running score through the Session Score: 0 Selected? false
Criterion: 81 Name: Simplify the main question squares
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: - Score: 0
Criterion: 82 Name: Hold the user's interest through the questionnaire
Metacriterion: 42 Name: Make the game interesting
Term: + Score: 0

Question: 136 Name: What are the components of the Session class? Parent Option: 0
Option: 249 Name: Start Time Score: 0 Selected? false
Option: 250 Name: End Time Score: 0 Selected? false
Option: 251 Name: Session Complete Indicator Score: 0 Selected? false
Option: 252 Name: User ID Score: 0 Selected? false

Question: 157 Name: Will partial plays be supported? Parent Option: 0
Option: 356 Name: Support partial plays Score: 0 Selected? false

Question: 170 Name: What is the purpose of the end message? Parent Option: 321
Option: 340 Name: Information to contextualise the questionnaire Score: 0 Selected? false

Outline: 146 Number: 06.0 Name: Square Path Master

Question: 188 Name: What is the relationship between SPM and Project? Parent Option: 0
Option: 373 Name: One Project, N SPMs for different groups Score: 0 Selected? false

Outline: 116 Number: 07.0 Name: User Square Path

Question: 52 Name: How will the User Square Path be constructed? Parent Option: 56
Option: 65 Name: Full path will be constructed for each user Score: 0 Selected? false
Criterion: 45 Name: Development time and cost
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 46 Name: System flexibility
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 47 Name: System performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Criterion: 52 Name: Enhance user navigation
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 66 Name: Path will be constructed dynamically for each user Score: 0 Selected? false
Criterion: 45 Name: Development time and cost
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 46 Name: System flexibility
Metacriterion: 30 Name: Flexibility of use

Term: + Score: 0
Criterion: 47 Name: System performance
Metacriterion: 33 Name: Maximise runtime performance
Term: - Score: 0
Criterion: 52 Name: Enhance user navigation
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0

Question: 71 Name: How are plan dependencies implemented? Parent Option: 0
Option: 101 Name: Full square path reduced by metrics Score: 0 Selected? false

Question: 85 Name: What are the attributes of User Square Path? Parent Option: 0
Option: 128 Name: Previous Square Score: 0 Selected? false
Option: 129 Name: Current Square Score: 0 Selected? false
Option: 130 Name: Next Square Score: 0 Selected? false
Option: 309 Name: Disabled Indicator Score: 0 Selected? false

Question: 87 Name: What does the User Square Path class do? Parent Option: 0
Option: 133 Name: Tells us how to get to the next Square Score: 0 Selected? false
Option: 255 Name: Starts as copy of Square Path Master and is reduced based on Metrics Score: 0 Selected? false

Question: 89 Name: How will Square dependencies be managed? Parent Option: 0
Option: 135 Name: A Metric can delete one Square Score: 0 Selected? false
Option: 136 Name: A Metric can delete multiple Squares Score: 0 Selected? false
Option: 137 Name: A Metric can deleted an entire Square Section Score: 0 Selected? false

Question: 137 Name: What is a User Square Path? Parent Option: 0
Option: 254 Name: User Square Path starts as a copy of Square Path Master Score: 0 Selected? false

Outline: 132 Number: 08.0 Name: Square Section

Question: 104 Name: What is the purpose of Square Section? Parent Option: 0
Option: 167 Name: Support disabling multiple Squares Score: 0 Selected? false
Criterion: 72 Name: Easier setup of questionnaire
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 76 Name: Easier depedency management
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Option: 183 Name: Group all dependent Squares in a Section Score: 0 Selected? false
Criterion: 72 Name: Easier setup of questionnaire
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 76 Name: Easier depedency management
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Option: 334 Name: Similar to a section of a questionnaire Score: 0 Selected? false
Criterion: 72 Name: Easier setup of questionnaire
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 76 Name: Easier depedency management
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0

Question: 150 Name: What are the attributes of Square Section? Parent Option: 0
Option: 299 Name: Square Section Name Score: 0 Selected? false

Outline: 93 Number: 09.0 Name: Square

Question: 35 Name: How many questions are supported on a square? Parent Option: 33
Option: 36 Name: One-to-one between square and questions Score: 0 Selected? false
Criterion: 56 Name: Ease of dependency maintenance
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Option: 37 Name: Multiple questions per square Score: 0 Selected? false
Criterion: 56 Name: Ease of dependency maintenance
Metacriterion: 43 Name: Easy to build questionnaire
Term: - Score: 0

Question: 36 Name: How will concepts or squares be ordered? Parent Option: 0
Option: 56 Name: Create a user square path or plan Score: 0 Selected? false

Option: 57 Name: Create a map for the user to navigate with Score: 0 Selected? false

Question: 46 Name: What are the UI components of a Square Parent Option: 33
Option: 50 Name: Overview text at the top of each Square Score: 0 Selected? false

Question: 50 Name: Will squares be organised in a hierarchy? Parent Option: 0
Option: 54 Name: Limit the hierarchy to 3 levels Score: 0 Selected? false
Criterion: 43 Name: Simpler to program
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 44 Name: More flexible, scalable
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 55 Name: Make the hierarchy completely dynamic Score: 0 Selected? false
Criterion: 43 Name: Simpler to program
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 44 Name: More flexible, scalable
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Question: 60 Name: How to manage skipped Squares? Parent Option: 0
Option: 84 Name: Provide a general score for skipped Squares Score: 0 Selected? false
Option: 85 Name: Provide a general "No Answer" facility Score: 0 Selected? false
Option: 86 Name: Provide a general "Don't Know" facility Score: 0 Selected? false
Option: 87 Name: Provide a general "Not applicable" facility Score: 0 Selected? false

Question: 70 Name: What are square dependencies? Parent Option: 0
Option: 98 Name: Squares exist in a sequence Score: 0 Selected? false
Option: 99 Name: Squares exist in a linked list Score: 0 Selected? false
Option: 100 Name: Metric has a plan override Score: 0 Selected? false

Question: 84 Name: How do we know what the first Square is? Parent Option: 0

Question: 101 Name: What are the attributes of a Square? Parent Option: 0
Option: 161 Name: Square Type Score: 0 Selected? false
Option: 266 Name: Navigation buttons Score: 0 Selected? false
Option: 271 Name: Current Square ID Score: 0 Selected? false
Option: 272 Name: Next Square ID Score: 0 Selected? false
Option: 297 Name: Scoring Group Score: 0 Selected? false

Question: 141 Name: What does Square do? Parent Option: 0
Option: 265 Name: Get the Questions Score: 0 Selected? false
Option: 270 Name: Get next Square Score: 0 Selected? false
Option: 286 Name: Make the call to evaluate the User Answers Score: 0 Selected? false

Question: 142 Name: What are the navigation buttons on a Square? Parent Option: 266
Option: 267 Name: Previous Score: 0 Selected? false
Option: 268 Name: Next Score: 0 Selected? false
Option: 269 Name: Home Score: 0 Selected? false

Outline: 126 Number: 10.0 Name: Question

Question: 47 Name: How will Questions be answered? Parent Option: 0
Option: 51 Name: Numeric format Score: 0 Selected? false
Criterion: 39 Name: Ease of data entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 52 Name: Choices (e.g. radio button, checkbox) Score: 0 Selected? false
Criterion: 39 Name: Ease of data entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0

Question: 58 Name: Should we provide help for Questions Parent Option: 0
Option: 77 Name: Use the overview text for the question help Score: 0 Selected? false

Question: 66 Name: How to re-do a Question Parent Option: 0

Question: 72 Name: How to maintain the Questions Parent Option: 0
Option: 102 Name: Track questions that are never visited Score: 0 Selected? false

Question: 80 Name: How are Questions answered? Parent Option: 0
Option: 116 Name: All Yes/No/No Answer Score: 0 Selected? false
Option: 117 Name: Allow numeric Answer Score: 0 Selected? false
Option: 118 Name: A numeric range Score: 0 Selected? false

Question: 93 Name: What are the components of a Question? Parent Option: 0
Option: 158 Name: Question Type Score: 0 Selected? false
Option: 296 Name: Question Weight Score: 0 Selected? false

Question: 111 Name: How to manage Question dependencies? Parent Option: 0
Option: 184 Name: Don't support dependencies Score: 0 Selected? false
Criterion: 78 Name: Dependencies represent knowledge in the domain
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 185 Name: Support dependencies Score: 0 Selected? false
Criterion: 78 Name: Dependencies represent knowledge in the domain
Metacriterion: 18 Name: Maximise educational value for users
Term: + Score: 0

Question: 112 Name: Create a new class for Question Dependency? Parent Option: 185

Question: 130 Name: How will sub-questions be handled? Parent Option: 0
Option: 228 Name: Create a new structure and functionality to support sub-questions Score: 0 Selected? false
Criterion: 115 Name: More natural, flexible questionnaire design
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0
Criterion: 116 Name: Simplicity of design/construction
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Option: 229 Name: User creates a separate question for each sub-question Score: 0 Selected? true
Criterion: 115 Name: More natural, flexible questionnaire design
Metacriterion: 30 Name: Flexibility of use
Term: - Score: 0
Criterion: 116 Name: Simplicity of design/construction
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 143 Name: What are the attributes of Question? Parent Option: 0
Option: 273 Name: ID Score: 0 Selected? false
Option: 274 Name: Question Text Score: 0 Selected? false
Option: 275 Name: Answer Control Score: 0 Selected? false
Option: 287 Name: Question weight Score: 0 Selected? false
Option: 349 Name: Display order Score: 0 Selected? false

Question: 144 Name: What class will get the Questions for a given Square? Parent Option: 0
Option: 276 Name: Question UI Score: 0 Selected? false
Criterion: 130 Name: Fewer host calls
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 277 Name: User Square Path Score: 0 Selected? false
Criterion: 130 Name: Fewer host calls
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 278 Name: Square Score: 0 Selected? false
Criterion: 130 Name: Fewer host calls
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Question: 148 Name: What answer controls are supported? Parent Option: 275
Option: 291 Name: Text field Score: 0 Selected? false
Option: 292 Name: Radio Button Score: 0 Selected? false
Option: 293 Name: Checkbox Score: 0 Selected? false
Option: 294 Name: Drop down list Score: 0 Selected? false

Question: 149 Name: How do we designate the a Question isn't scored independently? Parent Option: 0

Question: 155 Name: Can we vary the working of the Questionnaire? Parent Option: 0
Option: 306 Name: Yes Score: 0 Selected? false
Option: 307 Name: Use different path for different questionnaire wording Score: 0 Selected? false

Question: 163 Name: Can we support Question sub-text? Parent Option: 0
Option: 328 Name: Yes, just add subtext as a field and print in a smaller font Score: 0 Selected? false

Question: 169 Name: How will we provide help for Questions? Parent Option: 0
Option: 339 Name: On the introduction page Score: 0 Selected? false

Outline: 144 Number: 11.0 Name: Question Dependency

Question: 113 Name: What is the purpose of Question Dependency? Parent Option: 0

Question: 114 Name: What are the attributes of Question Dependency? Parent Option: 0
Option: 186 Name: Affecting Question Score: 0 Selected? false
Option: 187 Name: Affected Question Score: 0 Selected? false

Question: 118 Name: Should we have Question Dependencies? Parent Option: 0
Option: 193 Name: Yes Score: 0 Selected? false
Criterion: 83 Name: Ease of programming
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 84 Name: Ease of questionnaire creation
Metacriterion: 43 Name: Easy to build questionnaire
Term: - Score: 0
Criterion: 85 Name: More accurate, realistic scoring
Metacriterion: 18 Name: Maximise educational value for users
Term: + Score: 0
Criterion: 137 Name: Good enough scoring
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Option: 194 Name: No Score: 0 Selected? false
Criterion: 83 Name: Ease of programming
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 84 Name: Ease of questionnaire creation
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 85 Name: More accurate, realistic scoring
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 137 Name: Good enough scoring
Metacriterion: 43 Name: Easy to build questionnaire
Term: - Score: 0

Question: 119 Name: How will Question Dependencies be implemented? Parent Option: 0
Option: 195 Name: Always as simple relationship between two Questions Score: 0 Selected? false
Option: 298 Name: Questions may be chained Score: 0 Selected? false
Option: 301 Name: One User Answer can adjust the weights of many other Questions Score: 0 Selected? false

Question: 134 Name: Put related Questions on the same page Parent Option: 0

Question: 151 Name: Can we do question dependency management through weight adjustments? Parent Option: 193

Outline: 127 Number: 12.0 Name: Answer

Question: 79 Name: What is Answer Class Parent Option: 62
Option: 113 Name: There is an Answer instancfe for each possible Question answer Score: 0 Selected? false
Option: 114 Name: An other instance if the standard Answers don't cover it Score: 0 Selected? false
Option: 115 Name: Answer text Score: 0 Selected? false

Question: 92 Name: What answer formats are supported? Parent Option: 0
Option: 140 Name: Text choices Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 141 Name: User text Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 142 Name: Numeric Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 143 Name: Percentage Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 144 Name: Yes/No Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions

Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 369 Name: Memo (long text) Score: 0 Selected? false
Criterion: 124 Name: Supports sub-questions
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Question: 94 Name: What is the form of a skipped answer? Parent Option: 0
Option: 145 Name: Don't know Score: 0 Selected? false
Criterion: 65 Name: Support all controls (eg radio)
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0
Option: 146 Name: No answer Score: 0 Selected? false
Criterion: 65 Name: Support all controls (eg radio)
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0
Option: 147 Name: Leave blank Score: 0 Selected? false
Criterion: 65 Name: Support all controls (eg radio)
Metacriterion: 30 Name: Flexibility of use
Term: - Score: 0

Question: 95 Name: Allow answers to be skipped? Parent Option: 0
Option: 148 Name: Yes Score: 0 Selected? false
Criterion: 63 Name: Support user flow through the system
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0
Criterion: 64 Name: Make scoring and comparisons easier
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Option: 149 Name: No Score: 0 Selected? false
Criterion: 63 Name: Support user flow through the system
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: - Score: 0
Criterion: 64 Name: Make scoring and comparisons easier
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 120 Name: What are the attributes of Answer class? Parent Option: 0
Option: 196 Name: Answer Text Score: 0 Selected? false

Question: 131 Name: How do note skipped answers? Parent Option: 148
Option: 230 Name: Don't know Score: 0 Selected? false
Option: 231 Name: No answer Score: 0 Selected? false
Option: 232 Name: Not applicable Score: 0 Selected? false

Question: 145 Name: What does Answer class do? Parent Option: 0
Option: 279 Name: Get Answer Text Score: 0 Selected? false
Option: 280 Name: Get Metrics Score: 0 Selected? false

Question: 158 Name: How are Answer defaults determined? Parent Option: 0
Option: 317 Name: Just default to the first Answer Score: 0 Selected? false
Criterion: 139 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Option: 318 Name: Default to No Answer Score: 0 Selected? false
Criterion: 139 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 159 Name: Should the number of Answers be limited for a Question? Parent Option: 0
Option: 319 Name: No limit Score: 0 Selected? false
Criterion: 141 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 142 Name: Clean web user interface
Metacriterion: 40 Name: Make the user interface aesthetically pleasing
Term: - Score: 0
Option: 320 Name: Limit to one page without scrolling Score: 0 Selected? false
Criterion: 141 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 142 Name: Clean web user interface
Metacriterion: 40 Name: Make the user interface aesthetically pleasing
Term: + Score: 0

Question: 160 Name: How does the user change previously entered answers? Parent Option: 0
Option: 322 Name: Link back from the results page to change Score: 0 Selected? false
Option: 323 Name: Use back button Score: 0 Selected? false
Option: 324 Name: Disallow changes Score: 0 Selected? false

Question: 161 Name: Do we keep overwritten User Answers? Parent Option: 322
Option: 325 Name: Yes keep them Score: 0 Selected? false
Criterion: 144 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 145 Name: Flexibility
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 326 Name: Just overwrite them Score: 0 Selected? false
Criterion: 144 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 145 Name: Flexibility
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 327 Name: Project dependent Score: 0 Selected? false
Criterion: 144 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 145 Name: Flexibility
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0

Question: 162 Name: How do we handle two-part Answers? Parent Option: 0

Question: 164 Name: Support vertical and horizontal display of q=Answer lists? Parent Option: 0
Option: 329 Name: Yes support both views Score: 0 Selected? false
Criterion: 147 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Option: 330 Name: No, only vertical orientation Score: 0 Selected? false
Criterion: 147 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 165 Name: Support standard scales for Answers? Parent Option: 0
Option: 331 Name: Yes support standard scales Score: 0 Selected? false
Criterion: 148 Name: Redundant data entry
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Criterion: 149 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Option: 332 Name: No standard scales Score: 0 Selected? false
Criterion: 148 Name: Redundant data entry
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 149 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0

Question: 177 Name: Should the size of user answers be limited? Parent Option: 0
Option: 354 Name: No limit Score: 0 Selected? false
Criterion: 164 Name: System Flexibility
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 166 Name: System Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 355 Name: Limit to HTTP supported size Score: 0 Selected? false
Criterion: 164 Name: System Flexibility
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 166 Name: System Performance
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Outline: 130 Number: 13.0 Name: User Answer

Question: 102 Name: What is the purpose of User Answer? Parent Option: 0
Option: 162 Name: Record the actual value from the user Score: 0 Selected? false

Question: 103 Name: What are the attributes of User Answer? Parent Option: 0
Option: 165 Name: The Metric that was applied Score: 0 Selected? false
Option: 166 Name: The user's answer text Score: 0 Selected? false

Question: 110 Name: What User Answer formats are supported? Parent Option: 0
Option: 175 Name: Number ranges Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: + Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: - Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: + Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 176 Name: Numbers Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: + Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: - Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 177 Name: Text options Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 178 Name: Free text Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: + Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 179 Name: Choices (checkboxes) Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction

Term: + Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Option: 180 Name: Radio buttons Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Option: 241 Name: Optional user entry or checklists Score: 0 Selected? false
Criterion: 86 Name: Easier evaluation with Metric
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 87 Name: Scalability
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Criterion: 106 Name: Easier user entry
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 107 Name: More accurate evaluation
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Criterion: 122 Name: Complexity
Metacriterion: 30 Name: Flexibility of use
Term: - Score: 0

Question: 121 Name: Include links to advice sites on Answer pages? Parent Option: 0

Option: 199 Name: Yes Score: 0 Selected? false
Criterion: 94 Name: Improves usefulness of the game
Metacriterion: 18 Name: Maximise educational value for users
Term: + Score: 0

Option: 200 Name: No Score: 0 Selected? false
Criterion: 94 Name: Improves usefulness of the game
Metacriterion: 18 Name: Maximise educational value for users
Term: - Score: 0

Question: 133 Name: Support calculated answers? Parent Option: 0

Option: 242 Name: Spreadsheet-like function for complex answers Score: 0 Selected? false
Criterion: 123 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0

Option: 243 Name: Suggest users go to Excel for complex calculations, then enter Score: 0 Selected? false
Criterion: 123 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 146 Name: How do we evaluate the User Answers? Parent Option: 0

Option: 285 Name: Match Metrics to User Answers Score: 0 Selected? false

Option: 288 Name: Determine that a usable Answer has been provided. Score: 0 Selected? false

Question: 147 Name: How do we evaluate multiple User Answers to a single Question? Parent Option: 0

Question: 191 Name: How are User Answers validated? Parent Option: 0

Option: 378 Name: Support JavaScripting Score: 0 Selected? false

Outline: 123 Number: 14.0 Name: Metric

Question: 48 Name: How will metrics be implemented Parent Option: 49
Option: 53 Name: Initial questions set up respondent type Score: 0 Selected? false

Question: 59 Name: What is the structure of a Metric Parent Option: 0
Option: 78 Name: Score Score: 0 Selected? false
Option: 79 Name: Advice Score: 0 Selected? false
Option: 80 Name: Links Score: 0 Selected? false

Question: 76 Name: What does the metric do? Parent Option: 0
Option: 105 Name: Evaluates the Answer Score: 0 Selected? false
Option: 106 Name: Manages the Square Path dependencies Score: 0 Selected? false
Option: 375 Name: Categorises answers Score: 0 Selected? false

Question: 78 Name: What are the responsibilities of the Metric class? Parent Option: 0
Option: 134 Name: Evaluate a user answer Score: 0 Selected? false

Question: 88 Name: How does the system match Metric to Answer Parent Option: 0
Option: 163 Name: Uses ranges of value to match User Answer to Metric Score: 0 Selected? false
Option: 164 Name: Uses exact value to match User Value to Metric Score: 0 Selected? false

Question: 96 Name: What is the form of the Metric evaluation Parent Option: 0
Option: 151 Name: Separate opportunity and risk scores Score: 0 Selected? false
Option: 300 Name: Sum and opportunity score times question weight Score: 0 Selected? false

Question: 97 Name: Are opportunity and risk properties, or separate classes? Parent Option: 151
Option: 152 Name: Propertiesof Metric Score: 0 Selected? false
Criterion: 67 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 68 Name: Flexibility and power
Metacriterion: 30 Name: Flexibility of use
Term: - Score: 0
Option: 153 Name: Separate related classes Score: 0 Selected? false
Criterion: 67 Name: Simplicity
Metacriterion: 31 Name: Simplicity of construction
Term: - Score: 0
Criterion: 68 Name: Flexibility and power
Metacriterion: 30 Name: Flexibility of use
Term: + Score: 0

Question: 98 Name: What are the behaviours of Metric? Parent Option: 0
Option: 154 Name: Disable other Squares Score: 0 Selected? false
Option: 289 Name: Score the Question Score: 0 Selected? false

Question: 99 Name: How are Squares disabled by Metrics? Parent Option: 154

Question: 108 Name: How do we manage Question dependencies? Parent Option: 0

Question: 109 Name: How will Metrics score a Question? Parent Option: 0
Option: 172 Name: Score by individual Question Score: 0 Selected? false
Criterion: 90 Name: Ease of implementation
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 91 Name: Ease of use
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 92 Name: Real world applicability
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 173 Name: Score by Square Section Score: 0 Selected? false
Criterion: 90 Name: Ease of implementation
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 91 Name: Ease of use
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 92 Name: Real world applicability
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0

Option: 174 Name: Score by entire play Score: 0 Selected? false
Criterion: 90 Name: Ease of implementation
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 91 Name: Ease of use
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 92 Name: Real world applicability
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 182 Name: Calculate a scoring factor from real company data Score: 0 Selected? false
Criterion: 90 Name: Ease of implementation
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0
Criterion: 91 Name: Ease of use
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 92 Name: Real world applicability
Metacriterion: 18 Name: Maximise educational value for users
Term: Score: 0
Option: 198 Name: Use a five point scale, qualitative answers Score: 0 Selected? false
Criterion: 90 Name: Ease of implementation
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0
Criterion: 91 Name: Ease of use
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Criterion: 92 Name: Real world applicability
Metacriterion: 18 Name: Maximise educational value for users
Term: - Score: 0

Question: 124 Name: How will we determine weights for opportunity and risk scores? Parent Option: 0
Option: 244 Name: Test with real users to determine weights Score: 0 Selected? false

Question: 128 Name: What will be the scale for opportunity and risk score? Parent Option: 0
Option: 225 Name: 1-5 Score: 0 Selected? false
Option: 226 Name: 1-10 Score: 0 Selected? false

Question: 129 Name: How to calculate risk and opportunity scores with weighting? Parent Option: 0
Option: 227 Name: 10 = high opportunity, 10 = low risk Score: 0 Selected? false
Criterion: 114 Name: Simple calculation
Metacriterion: 31 Name: Simplicity of construction
Term: + Score: 0

Question: 190 Name: How are risk and opportunity scores used? Parent Option: 0
Option: 376 Name: Properties of Metric Score: 0 Selected? false
Option: 377 Name: Allow positive and negative offsetting assessments Score: 0 Selected? false

Outline: 133 Number: 15.0 Name: Path Update

Question: 69 Name: How is the user path updated? Parent Option: 0
Option: 308 Name: Simply flag a square as skipped Score: 0 Selected? false

Question: 75 Name: How are the path dependencies created and maintained Parent Option: 103
Option: 104 Name: All Squares are initially 'on' then are checked off Score: 0 Selected? false
Criterion: 54 Name: Simpler to program
Metacriterion: 31 Name: Simplicity of construction
Term: Score: 0

Question: 105 Name: What is the purpose of Path Update? Parent Option: 0
Option: 295 Name: Manage the dependencies between Squares Score: 0 Selected? false

Question: 106 Name: What are the attributes of Path Update? Parent Option: 0
Option: 168 Name: Square to skip Score: 0 Selected? false
Option: 169 Name: Square Section to skip Score: 0 Selected? false

Outline: 149 Number: 16.0 Name: Project

Outline: 148 Number: 17.0 Name: Project Groups

Question: 152 Name: What are the attributes of Project Group? Parent Option: 0
Option: 302 Name: Target Group Size Score: 0 Selected? false

Question: 153 Name: What are the behaviours of Project Group? Parent Option: 0
Option: 303 Name: Assign User to a Group Score: 0 Selected? false
Option: 304 Name: Manage Group assignment Score: 0 Selected? false
Option: 305 Name: Randomize Users into Groups Score: 0 Selected? false

Question: 154 Name: How will group sizes be normalized? Parent Option: 303

Question: 189 Name: Allow project groups to be non-random? Parent Option: 0
Option: 374 Name: Yes Score: 0 Selected? false

Outline: 150 Number: 18.0 Name: vqServlet

Question: 156 Name: What are the behaviours of the vqServlet main class? Parent Option: 0
Option: 310 Name: Check for duplicate User for the Project Score: 0 Selected? false
Option: 311 Name: Create the User Score: 0 Selected? false
Option: 312 Name: Create the Session Score: 0 Selected? false

Outline: 151 Number: 19.0 Name: Export

Question: 173 Name: What is the purpose of the Export class? Parent Option: 0
Option: 346 Name: Utility class for exporg to statistics packages Score: 0 Selected? false

Outline: 154 Number: 40.0 Name: Static HTML Start Page

Question: 187 Name: What is the purpose of the static HTML start page? Parent Option: 0
Option: 372 Name: Calls the Start Servlet Score: 0 Selected? false

Outline: 152 Number: 50.0 Name: General User Interface

Question: 74 Name: What is the user interface for game creation Parent Option: 0
Option: 103 Name: A series of tables Score: 0 Selected? false

Question: 90 Name: How do we facilitate user navigation Parent Option: 0
Option: 138 Name: Provide a Square Path map and let the user choose Score: 0 Selected? false
Criterion: 58 Name: User control
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0
Criterion: 59 Name: User sense of place
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0
Criterion: 155 Name: Provide an indicator of where users are in the hierarchy
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 139 Name: Provide initial set of Squares that sets up the rest of the Path Score: 0 Selected? false
Criterion: 58 Name: User control
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 59 Name: User sense of place
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 155 Name: Provide an indicator of where users are in the hierarchy
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0

Question: 91 Name: What is the graphic design template for the pages? Parent Option: 0
Option: 207 Name: Minimalist Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 208 Name: Flashy, with animation (eg use Flash) Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 209 Name: Include project description Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 210 Name: Map graphic with rollover text Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0
Option: 211 Name: Mission statement Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance

Term: Score: 0
Option: 221 Name: Drop-down navigation menu Score: 0 Selected? false
Criterion: 103 Name: Page load time
Metacriterion: 33 Name: Maximise runtime performance
Term: Score: 0

Question: 107 Name: Do we base the UI on the use of tabs for Square Sections? Parent Option: 0
Option: 170 Name: No Score: 0 Selected? false
Criterion: 73 Name: Familiar navigation mechanism
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 74 Name: Not scalable as the number of sections grows
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 171 Name: Yes Score: 0 Selected? false
Criterion: 73 Name: Familiar navigation mechanism
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 74 Name: Not scalable as the number of sections grows
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0

Question: 123 Name: What is the mechanism for user navigation? Parent Option: 0
Option: 212 Name: Frames Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: - Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 213 Name: Style Sheets Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: + Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: + Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 214 Name: Table with navigation bar Score: 0 Selected? true
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 215 Name: Dynamically generate navigation aids with Java Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 238 Name: Tabs Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: - Score: 0
Option: 239 Name: Drop Down menu Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard

Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: Score: 0
Option: 240 Name: Rollover map Score: 0 Selected? false
Criterion: 104 Name: Conform to hypertext standard
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Criterion: 105 Name: Easy to create and maintain
Metacriterion: 43 Name: Easy to build questionnaire
Term: Score: 0
Criterion: 119 Name: Scalable
Metacriterion: 36 Name: Use scalable user interface components
Term: - Score: 0

Question: 132 Name: Orientation and location of navigation bar Parent Option: 214
Option: 236 Name: Veritical Score: 0 Selected? false
Option: 237 Name: Horizontal Score: 0 Selected? false

Question: 174 Name: Show record IDs? Parent Option: 0
Option: 347 Name: No Score: 0 Selected? false
Criterion: 156 Name: Screen clutter
Metacriterion: 40 Name: Make the user interface aesthetically pleasing
Term: Score: 0
Criterion: 157 Name: Useful information for users
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0
Option: 348 Name: Yes Score: 0 Selected? false
Criterion: 156 Name: Screen clutter
Metacriterion: 40 Name: Make the user interface aesthetically pleasing
Term: Score: 0
Criterion: 157 Name: Useful information for users
Metacriterion: 42 Name: Make the game interesting
Term: Score: 0

Question: 175 Name: Use style sheets for the HTML interface? Parent Option: 0
Option: 350 Name: Yes Score: 0 Selected? false
Criterion: 160 Name: Flexible UI and visual creation
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Option: 351 Name: No Score: 0 Selected? false
Criterion: 160 Name: Flexible UI and visual creation
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0

Question: 176 Name: In what format should all user text be entered? Parent Option: 0
Option: 352 Name: HTML Score: 0 Selected? false
Criterion: 161 Name: Flexible, allows embedded links
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 162 Name: Simpler for users to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0
Option: 353 Name: ASCII Score: 0 Selected? false
Criterion: 161 Name: Flexible, allows embedded links
Metacriterion: 30 Name: Flexibility of use
Term: Score: 0
Criterion: 162 Name: Simpler for users to understand
Metacriterion: 45 Name: Easy to use questionnaire (respondent)
Term: Score: 0

Question: 179 Name: Include an advanced mode for more complex features? Parent Option: 0
Option: 357 Name: Include advanced mode Score: 0 Selected? false

Question: 186 Name: How to make meaning of defaults clear to users? Parent Option: 0

Outline: 153 Number: 60.0 Name: Security

Question: 65 Name: How do we handle privacy Parent Option: 0
Option: 95 Name: Provide the privacy policy in the Help system Score: 0 Selected? false

Question: 181 Name: How will projects be protected? Parent Option: 0

Question: 182 Name: What are the levels of security? Parent Option: 0
Option: 358 Name: Users Score: 0 Selected? false
Option: 359 Name: Super users Score: 0 Selected? false
Option: 360 Name: Administrators Score: 0 Selected? false

Question: 183 Name: What can users do? Parent Option: 358
Option: 361 Name: Read individual projects Score: 0 Selected? false
Option: 362 Name: Edit individual projects Score: 0 Selected? false
Option: 363 Name: Not create or delete projects Score: 0 Selected? false

Question: 184 Name: What can administrators do? Parent Option: 0
Option: 364 Name: All user privs Score: 0 Selected? false
Option: 365 Name: Create projects Score: 0 Selected? false

Question: 185 Name: What can supers do? Parent Option: 0
Option: 366 Name: Full privs on all projects Score: 0 Selected? false

Outline: 88 Number: 91.0 Name: System Development

Outline: 50 Number: 92.0 Name: System Testing

Question: 42 Name: Who will test the software? Parent Option: 0
Option: 44 Name: Actual potential users i.e. entrepreneurs Score: 0 Selected? false
Option: 48 Name: Put it on the web and solicit feedback Score: 0 Selected? false
Option: 197 Name: Venture capitalists Score: 0 Selected? false

Question: 43 Name: How will we recruit testers? Parent Option: 44
Option: 45 Name: Attend First Tuesday Score: 0 Selected? false

Question: 178 Name: Will load testing be conducted? Parent Option: 0

Outline: 48 Number: 99.0 Name: Administration

Question: 38 Name: Will the system be copyrighted? Parent Option: 0

Question: 39 Name: Who will own the rights to the system? Parent Option: 0
Option: 40 Name: London School of Economics Score: 0 Selected? false
Option: 46 Name: The LSE IS Department Score: 0 Selected? false